Sessions and Pipelines for Structured Service Programming

Michele Boreale[1] Roberto Bruni[2]
Rocco De Nicola[1] Michele Loreti[1]

[1]Dipartimento di Sistemi ed Informatica
Università di Firenze

[2]Dipartimento di Informatica
Università di Pisa

FMOODS 2008
Oslo, Norway
June 5th, 2008

Outline

Service Oriented Computing (SOC)

Services

SOC 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.

e-Expectations

Big companies put many efforts in promoting service delivery on a variety of computing platforms.
Tomorrow, there will be a plethora of new services for e-government, e-business, and e-health, and others within the rapidly evolving Information Society.

A crucial fact

Industrial consortia are developing orchestration and choreography languages, targeting the standardization of Web Services and XML-centric technologies, but *they lack neat semantic foundations.*

Service Oriented Computing (SOC)

Services

SOC 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.

e-Expectations

Big companies put many efforts in promoting service delivery on a variety of computing platforms.
Tomorrow, there will be a plethora of new services for e-government, e-business, and e-health, and others within the rapidly evolving Information Society.

A crucial fact

Industrial consortia are developing orchestration and choreography languages, targeting the standardization of Web Services and XML-centric technologies, but *they lack neat semantic foundations*.

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

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

Aim

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

Strategy

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

The role of process calculi

Coordinating and combining services

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

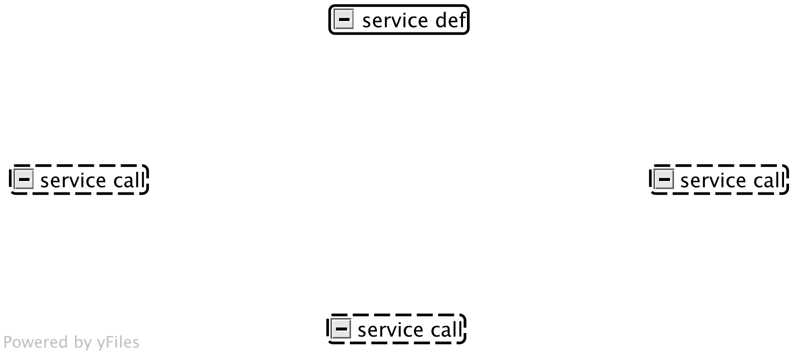
Sensoria workpackage 2

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

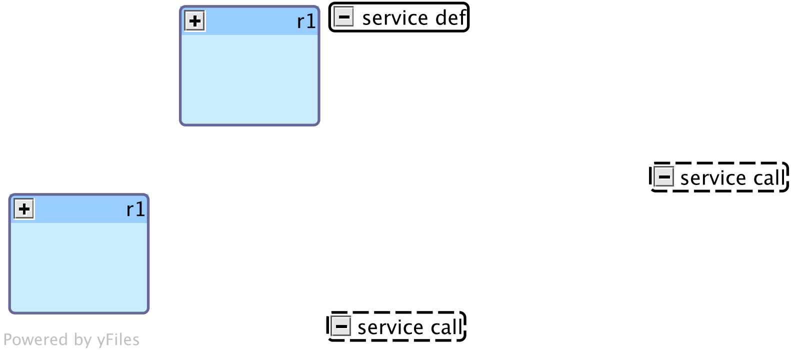
Sensoria core calculi

- *Signal Calculus*: middleware level
- *SOCK, COWS*: service level, correlation-based
- *SCC-family (SCC, SSCC, CC,* **CaSPiS***)*: service level, session-based
- *cc-pi, lambda-req*: SLA contract level

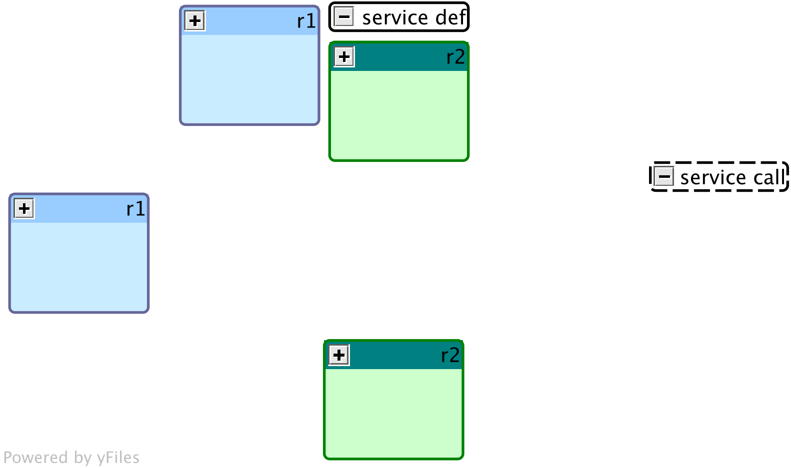
Sketch of Multiple Sessions



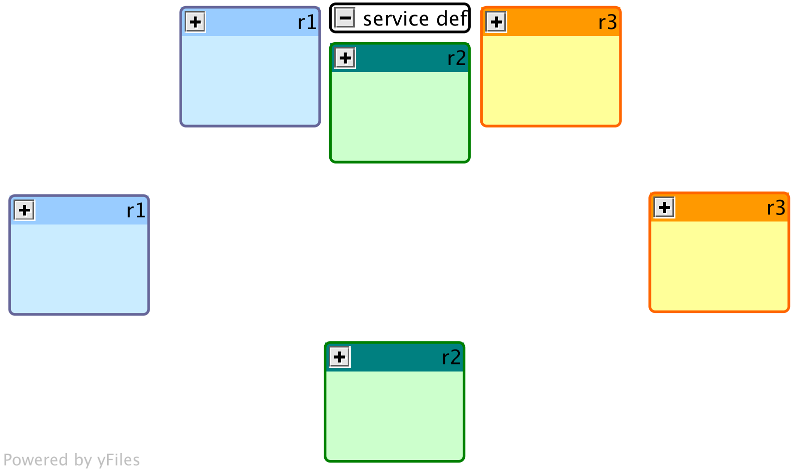
Sketch of Multiple Sessions



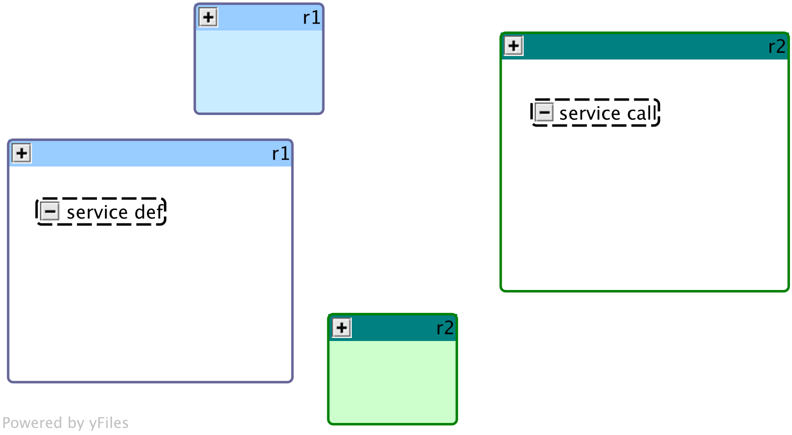
Sketch of Multiple Sessions



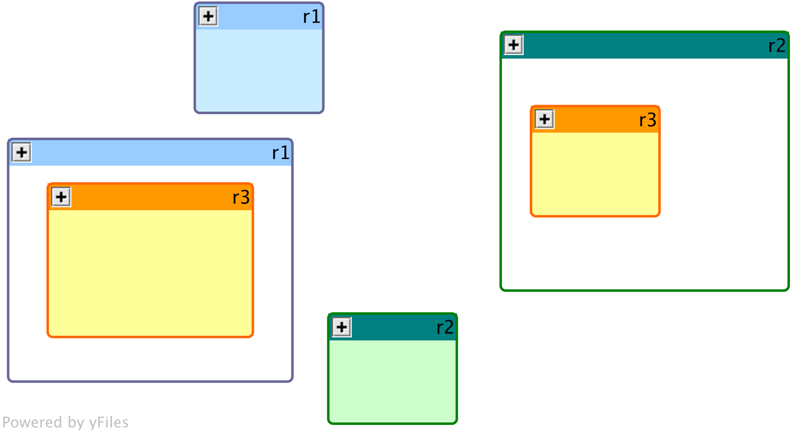
Sketch of Multiple Sessions



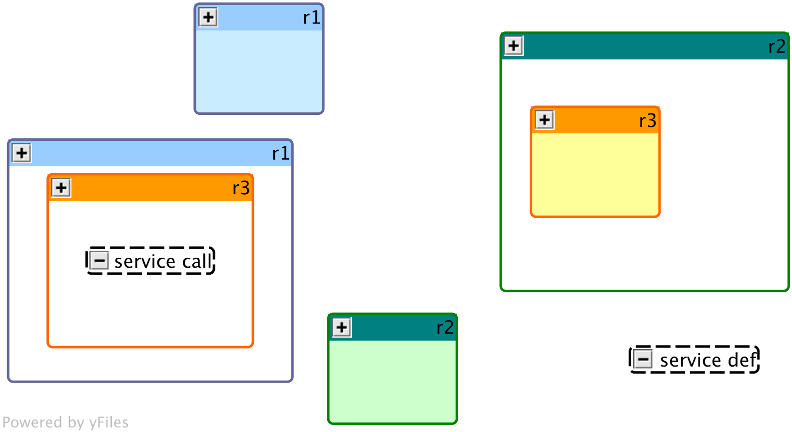
Sketch of Nested Sessions



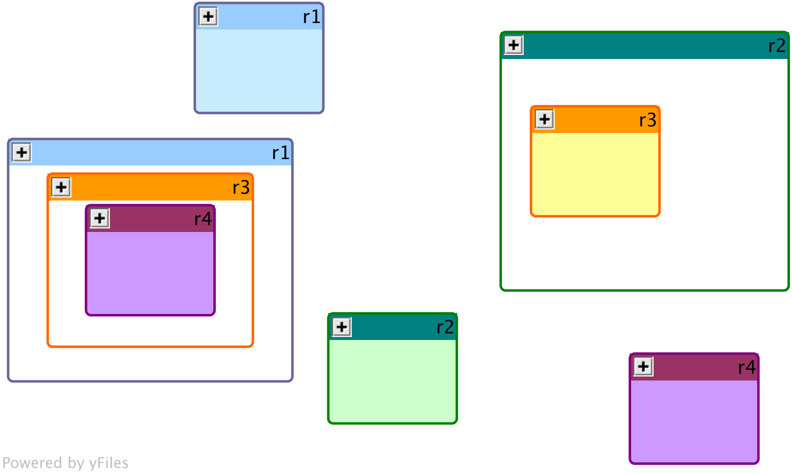
Sketch of Nested Sessions



Sketch of Nested Sessions



Sketch of Nested Sessions



Main Contribution

Proceedings

- Syntax + LTS semantics + reduction semantics (see Lemma 3)
- Basics of the language by several simple examples
- Flexibility of the language by a couple more sophisticated examples
- **Graceful termination of (nested) sessions**: We define a class of processes, called *balanced*, for which we can guarantee that no session-side is forced to hang forever after the abandon of its partner (see Theorem 1).

Talk

- Sketches of Syntax + Semantics by examples
- Balanced processes, informally + Graceful termination by examples

Disclaim

To keep in mind

We are dealing with conceptual abstractions: the syntax does not necessarily expose implementation details.

Examples

- A session is a logical entity that can be implemented by an additional *sid* parameter carried by all related messaging
- All service instances (serving different requests) can be handled by one service port

Outline

CaSPiS Genesis

Sources of inspiration

SCC [WS-FM 2006] was inspired by:

- π (names, communication): $x(y).P$, $\overline{x}y.P$, $(\nu x)P$
- πI, session types (primitives for sessions): $a(k).P$, $\overline{a}(k).P$ (roughly, think of $\overline{a}(k).P$ as $(\nu k)\overline{a}k.P$)
- Orc (pipelining and pruning of activities): $(EAPLS\langle 2008\rangle \mid EATCS\langle 2008\rangle) > cfp > Email\langle \mathtt{rb@gmail.it}, cfp\rangle$ $Email\langle \mathtt{rb@gmail.it}, cfp\rangle$ **where** $cfp :\in (EAPLS\langle 2008\rangle \mid EATCS\langle 2008\rangle)$

CaSPiS is inspired by SCC and:

- webπ, cjoin, Sagas (primitives for LRT and compensations)
- KLAIM (pattern matching)

All source were relevant to the SOC paradigm, but so far

- not available in a single calculus
- yet to be amalgamated in some disciplined way

CaSPiS Genesis

Sources of inspiration

SCC [WS-FM 2006] was inspired by:

- π (names, communication): $x(y).P$, $\overline{x}y.P$, $(\nu x)P$
- πI, session types (primitives for sessions): $a(k).P$, $\overline{a}(k).P$
  (roughly, think of $\overline{a}(k).P$ as $(\nu k)\overline{a}k.P$)
- Orc (pipelining and pruning of activities):
  $(\mathit{EAPLS}\langle 2008\rangle \mid \mathit{EATCS}\langle 2008\rangle) > \mathit{cfp} > \mathit{Email}\langle \texttt{rb@gmail.it}, \mathit{cfp}\rangle$
  $\mathit{Email}\langle \texttt{rb@gmail.it}, \mathit{cfp}\rangle$ **where** $\mathit{cfp} :\in (\mathit{EAPLS}\langle 2008\rangle \mid \mathit{EATCS}\langle 2008\rangle)$

CaSPiS is inspired by SCC and:

- webπ, cjoin, Sagas (primitives for LRT and compensations)
- KLAIM (pattern matching)

All source were relevant to the SOC paradigm, but so far

- not available in a single calculus
- yet to be amalgamated in some disciplined way

CaSPiS Genesis

Sources of inspiration

SCC [WS-FM 2006] was inspired by:

- π (names, communication): $x(y).P$, $\overline{x}y.P$, $(\nu x)P$
- πI, session types (primitives for sessions): $a(k).P$, $\overline{a}(k).P$ (roughly, think of $\overline{a}(k).P$ as $(\nu k)\overline{a}k.P$)
- Orc (pipelining and pruning of activities):
  $\big(EAPLS\langle 2008\rangle \mid EATCS\langle 2008\rangle\big) > cfp > Email\langle \texttt{rb@gmail.it}, cfp\rangle$
  $Email\langle \texttt{rb@gmail.it}, cfp\rangle$ **where** $cfp :\in \big(EAPLS\langle 2008\rangle \mid EATCS\langle 2008\rangle\big)$

CaSPiS is inspired by SCC and:

- webπ, cjoin, Sagas (primitives for LRT and compensations)
- KLAIM (pattern matching)

All source were relevant to the SOC paradigm, but so far

- not available in a single calculus
- yet to be amalgamated in some disciplined way

CaSPiS Genesis

Sources of inspiration

SCC [WS-FM 2006] was inspired by:

- π (names, communication): $x(y).P$, $\overline{x}y.P$, $(\nu x)P$
- πI, session types (primitives for sessions): $a(k).P$, $\overline{a}(k).P$ (roughly, think of $\overline{a}(k).P$ as $(\nu k)\overline{a}k.P$)
- Orc (pipelining and pruning of activities):
  $\big(EAPLS\langle 2008\rangle \mid EATCS\langle 2008\rangle\big) > cfp > Email\langle \texttt{rb@gmail.it}, cfp\rangle$
  $Email\langle \texttt{rb@gmail.it}, cfp\rangle$ **where** $cfp :\in \big(EAPLS\langle 2008\rangle \mid EATCS\langle 2008\rangle\big)$

CaSPiS is inspired by SCC and:

- webπ, cjoin, Sagas (primitives for LRT and compensations)
- KLAIM (pattern matching)

All source were relevant to the SOC paradigm, but so far

- not available in a single calculus
- yet to be amalgamated in some disciplined way

CaSPiS: General Principles

Service definitions: $s.P$

- services expose their protocols
- services can be deployed dynamically, shut down and updated
- services can handle multiple requests separately

Service invocations: $\overline{s}.P$

- service invocations expose their protocols
- sequential composition via pipelining (á la Orc)

Sessions: $r \triangleright P$

- service invocation spawns fresh session parties (locally to each partner)
- sessions are: two-party (service-side + client-side) + private
- interaction between session protocols: bi-directional
- nested sessions: values can be returned outside sessions (one level up)

CaSPiS: General Principles

Service definitions: $s.P$

- services expose their protocols
- services can be deployed dynamically, shut down and updated
- services can handle multiple requests separately

Service invocations: $\overline{s}.P$

- service invocations expose their protocols
- sequential composition via pipelining (á la Orc)

Sessions: $r \rhd P$

- service invocation spawns fresh session parties (locally to each partner)
- sessions are: two-party (service-side + client-side) + private
- interaction between session protocols: bi-directional
- nested sessions: values can be returned outside sessions (one level up)

CaSPiS: General Principles

Service definitions: $s.P$

- services expose their protocols
- services can be deployed dynamically, shut down and updated
- services can handle multiple requests separately

Service invocations: $\overline{s}.P$

- service invocations expose their protocols
- sequential composition via pipelining (á la Orc)

Sessions: $r \triangleright P$

- service invocation spawns fresh session parties (locally to each partner)
- sessions are: two-party (service-side + client-side) + private
- interaction between session protocols: bi-directional
- nested sessions: values can be returned outside sessions (one level up)

CaSPiS Syntax

Prefixes, Values, Patterns

$$
\begin{array}{rcll}
\pi & ::= & (F) & \text{Abstraction} \\
 & | & \langle V \rangle & \text{Concretion} \\
 & | & \langle V \rangle^{\uparrow} & \text{Return} \\
 & & & \\
V & ::= & u \mid f(\tilde{V}) & \text{Value } (f \in \Sigma) \\
 & & & \\
F & ::= & u \mid ?x \mid f(\tilde{F}) & \text{Pattern } (f \in \Sigma)
\end{array}
$$

Processes

$$
\begin{array}{rcllcll}
P, Q & ::= & \sum_{i \in I} \pi_i P_i & \text{Guarded Sum} & | & \dagger(k) & \text{Signal} \\
 & | & s_k.P & \text{Service Definition} & | & r \rhd_k P & \text{Session} \\
 & | & \bar{s}_k.P & \text{Service Invocation} & | & \blacktriangleright P & \text{Terminated Session} \\
 & | & P > Q & \text{Pipeline} & | & P|Q & \text{Parallel Composition} \\
 & | & \mathsf{close} & \text{Close} & | & (\nu n)P & \text{Restriction} \\
 & | & k \cdot P & \text{Listener} & | & !P & \text{Replication}
\end{array}
$$

CaSPiS Syntax

Prefixes, Values, Patterns

$$\begin{array}{rcll} \pi & ::= & (F) & \text{Abstraction} \\ & \mid & \langle V \rangle & \text{Concretion} \\ & \mid & \langle V \rangle^{\uparrow} & \text{Return} \end{array}$$

$$V \;::=\; u \mid f(\tilde{V}) \qquad \text{Value } (f \in \Sigma)$$

$$F \;::=\; u \mid ?x \mid f(\tilde{F}) \qquad \text{Pattern } (f \in \Sigma)$$

Processes

$$\begin{array}{rcllcll} P, Q & ::= & \sum_{i \in I} \pi_i P_i & \text{Guarded Sum} & \mid & \dagger(k) & \text{Signal} \\ & \mid & s_k.P & \text{Service Definition} & \mid & r \rhd_k P & \text{Session} \\ & \mid & \overline{s}_k.P & \text{Service Invocation} & \mid & \blacktriangleright P & \text{Terminated Session} \\ & \mid & P > Q & \text{Pipeline} & \mid & P \mid Q & \text{Parallel Composition} \\ & \mid & \mathsf{close} & \text{Close} & \mid & (\nu n)P & \text{Restriction} \\ & \mid & k \cdot P & \text{Listener} & \mid & !P & \text{Replication} \end{array}$$

Example 1: Digital Documents

Service definition

$$!\mathtt{sign}.(?x)(\nu t)\langle K\{x,t\}\rangle$$

- `sign` is a (replicated and thus persistent) service
- a `sign` instance waits for a digital document x, generates a fresh nonce t and then sends back both the document and the nonce signed with a key K

Service invocation

$$\overline{\mathtt{sign}}.\langle \text{plan}\rangle(?y)\langle y\rangle^{\uparrow}$$

- a client of `sign`
- it passes the argument plan to the service, then waits for the signed response from the server and returns this value outside the session as a result

Example 1: Digital Documents

Service definition

$$!\mathtt{sign}.(?x)(\nu t)\langle K\{x,t\}\rangle$$

- $\mathtt{sign}$ is a (replicated and thus persistent) service
- a $\mathtt{sign}$ instance waits for a digital document x, generates a fresh nonce t and then sends back both the document and the nonce signed with a key K

Service invocation

$$\overline{\mathtt{sign}}.\langle \mathsf{plan}\rangle(?y)\langle y\rangle^{\uparrow}$$

- a client of $\mathtt{sign}$
- it passes the argument plan to the service, then waits for the signed response from the server and returns this value outside the session as a result

Example 1: Digital Documents

A run

$$
\begin{array}{lll}
!\mathtt{sign}.(?x)(\nu t)\langle K\{x,t\}\rangle & & \mid \overline{\mathtt{sign}}.\langle \mathsf{plan}\rangle(?y)\langle y\rangle^{\uparrow} \\
!\mathtt{sign}.(?x)(\nu t)\langle K\{x,t\}\rangle & \mid (\nu r)\big(\ r \rhd (?x)(\nu t)\langle K\{x,t\}\rangle & \mid r \rhd \langle \mathsf{plan}\rangle(?y)\langle y\rangle^{\uparrow}\ \big) \\
!\mathtt{sign}.(?x)(\nu t)\langle K\{x,t\}\rangle & \mid (\nu r,t)\big(\ r \rhd \langle K\{\mathsf{plan},t\}\rangle & \mid r \rhd (?y)\langle y\rangle^{\uparrow}\ \big) \\
!\mathtt{sign}.(?x)(\nu t)\langle K\{x,t\}\rangle & \mid (\nu r,t)\big(\ r \rhd \mathbf{0} & \mid r \rhd \langle K\{\mathsf{plan},t\}\rangle^{\uparrow}\ \big)
\end{array}
$$

Sessions for separation

$$\big(\ \overline{\mathtt{sign}}.\langle \mathsf{plan}_1\rangle(?y)\langle y\rangle^{\uparrow} \quad \mid \quad \overline{\mathtt{sign}}.\langle \mathsf{plan}_2\rangle(?y)\langle y\rangle^{\uparrow}\ \big)$$

The protocols of the two clients will run in separate sessions and will not interfere.

Pipelines for composition

$$\big(\ \overline{\mathtt{sign}}.\langle \mathsf{plan}_1\rangle(?y)\langle y\rangle^{\uparrow} \quad \mid \quad \overline{\mathtt{sign}}.\langle \mathsf{plan}_2\rangle(?y)\langle y\rangle^{\uparrow}\ \big) \quad > \quad (?z)\overline{\mathtt{store}}.\langle z\rangle$$

Example 1: Digital Documents

A run

$$
\begin{array}{lll}
!\mathtt{sign}.(?x)(\nu t)\langle K\{x,t\}\rangle & & \mid \overline{\mathtt{sign}}.\langle \mathsf{plan}\rangle(?y)\langle y\rangle^{\uparrow} \\
!\mathtt{sign}.(?x)(\nu t)\langle K\{x,t\}\rangle & \mid (\nu r)(\ r \rhd (?x)(\nu t)\langle K\{x,t\}\rangle & \mid r \rhd \langle \mathsf{plan}\rangle(?y)\langle y\rangle^{\uparrow}\ ) \\
!\mathtt{sign}.(?x)(\nu t)\langle K\{x,t\}\rangle & \mid (\nu r,t)(\ r \rhd \langle K\{\mathsf{plan},t\}\rangle & \mid r \rhd (?y)\langle y\rangle^{\uparrow}\ ) \\
!\mathtt{sign}.(?x)(\nu t)\langle K\{x,t\}\rangle & \mid (\nu r,t)(\ r \rhd \mathbf{0} & \mid r \rhd \langle K\{\mathsf{plan},t\}\rangle^{\uparrow}\ )
\end{array}
$$

Sessions for separation

$$
(\ \overline{\mathtt{sign}}.\langle \mathsf{plan}_1\rangle(?y)\langle y\rangle^{\uparrow} \quad \mid \quad \overline{\mathtt{sign}}.\langle \mathsf{plan}_2\rangle(?y)\langle y\rangle^{\uparrow}\ )
$$

The protocols of the two clients will run in separate sessions and will not interfere.

Pipelines for composition

$$
(\ \overline{\mathtt{sign}}.\langle \mathsf{plan}_1\rangle(?y)\langle y\rangle^{\uparrow} \quad \mid \quad \overline{\mathtt{sign}}.\langle \mathsf{plan}_2\rangle(?y)\langle y\rangle^{\uparrow}\ ) \quad > \quad (?z)\overline{\mathtt{store}}.\langle z\rangle
$$

Example 1: Digital Documents

A run

$$
\begin{array}{lll}
!\mathtt{sign}.(?x)(\nu t)\langle K\{x,t\}\rangle & & \mid \overline{\mathtt{sign}}.\langle \mathsf{plan}\rangle(?y)\langle y\rangle^{\uparrow} \\
!\mathtt{sign}.(?x)(\nu t)\langle K\{x,t\}\rangle & \mid (\nu r)\big(\ r \rhd (?x)(\nu t)\langle K\{x,t\}\rangle & \mid r \rhd \langle \mathsf{plan}\rangle(?y)\langle y\rangle^{\uparrow}\ \big) \\
!\mathtt{sign}.(?x)(\nu t)\langle K\{x,t\}\rangle & \mid (\nu r,t)\big(\ r \rhd \langle K\{\mathsf{plan},t\}\rangle & \mid r \rhd (?y)\langle y\rangle^{\uparrow}\ \big) \\
!\mathtt{sign}.(?x)(\nu t)\langle K\{x,t\}\rangle & \mid (\nu r,t)\big(\ r \rhd \mathbf{0} & \mid r \rhd \langle K\{\mathsf{plan},t\}\rangle^{\uparrow}\ \big)
\end{array}
$$

Sessions for separation

$$\big(\ \overline{\mathtt{sign}}.\langle \mathsf{plan}_1\rangle(?y)\langle y\rangle^{\uparrow} \quad \mid \quad \overline{\mathtt{sign}}.\langle \mathsf{plan}_2\rangle(?y)\langle y\rangle^{\uparrow}\ \big)$$

The protocols of the two clients will run in separate sessions and will not interfere.

Pipelines for composition

$$\big(\ \overline{\mathtt{sign}}.\langle \mathsf{plan}_1\rangle(?y)\langle y\rangle^{\uparrow} \quad \mid \quad \overline{\mathtt{sign}}.\langle \mathsf{plan}_2\rangle(?y)\langle y\rangle^{\uparrow}\ \big) \quad > \quad (?z)\overline{\mathtt{store}}.\langle z\rangle$$

Example 2: Common Patterns of Interaction

One way

$$s.(?x) \qquad \bar{s}.\langle V \rangle$$

Request response

$$s.(?x)\langle f(x)\rangle \qquad \bar{s}.\langle V\rangle(?r)\langle r\rangle^{\uparrow}$$

π-calculus channels

$$a(x).P \triangleq a.(?x)\langle x\rangle^{\uparrow} > (?x)P \qquad \bar{a}v.P \triangleq \bar{a}.\langle v\rangle\langle -\rangle^{\uparrow} > (-)P$$

Proxy (service name passing)

$$!proxy.(?s, ?x)\bar{s}.\langle x\rangle!(?y)\langle y\rangle^{\uparrow}$$

Example 2: Common Patterns of Interaction

One way

$$s.(?x) \qquad \overline{s}.\langle V \rangle$$

Request response

$$s.(?x)\langle f(x)\rangle \qquad \overline{s}.\langle V\rangle(?r)\langle r\rangle^{\uparrow}$$

π-calculus channels

$$a(x).P \triangleq a.(?x)\langle x\rangle^{\uparrow} > (?x)P \qquad \overline{a}v.P \triangleq \overline{a}.\langle v\rangle\langle -\rangle^{\uparrow} > (-)P$$

Proxy (service name passing)

$$!proxy.(?s,?x)\overline{s}.\langle x\rangle!(?y)\langle y\rangle^{\uparrow}$$

Example 2: Common Patterns of Interaction

One way

$$s.(?x) \qquad \overline{s}.\langle V\rangle$$

Request response

$$s.(?x)\langle f(x)\rangle \qquad \overline{s}.\langle V\rangle(?r)\langle r\rangle^{\uparrow}$$

π-calculus channels

$$a(x).P \triangleq a.(?x)\langle x\rangle^{\uparrow} > (?x)P \qquad \overline{a}v.P \triangleq \overline{a}.\langle v\rangle\langle -\rangle^{\uparrow} > (-)P$$

Proxy (service name passing)

$$!proxy.(?s, ?x)\overline{s}.\langle x\rangle!(?y)\langle y\rangle^{\uparrow}$$

Example 3: Selection

Select

$$\textbf{select } F_1, \ldots, F_n \textbf{ from } P \triangleq (\nu s)\left(s.(F_1).\ldots.(F_n)\langle F_1^{-?}, \ldots, F_n^{-?}\rangle^{\uparrow} \mid \overline{s}.P\right)$$

where $F_i^{-?}$ denotes the value V_i obtained from F_i by replacing each $?x$ with x

Select-from

$$\textbf{select } F_1, \ldots, F_n \textbf{ from } P \textbf{ in } Q \triangleq \textbf{select } F_1, \ldots, F_n \textbf{ from } P > (F_1, \ldots, F_n)Q$$

Select first two CfP

$$\textbf{select } ?x, ?y \textbf{ from } \left(\overline{\texttt{EAPLS}}^* \mid \overline{\texttt{EATCS}}^* \mid \overline{\texttt{TYPES}}^*\right) \textbf{ in } \overline{\texttt{emailMe}}.\langle x, y\rangle$$

where

$$\overline{s}^* \triangleq \overline{s}.!(?x)\langle x\rangle^{\uparrow}$$

Outline

CaSPiS: Advanced Principles

Service definitions: $s_k.P$, $k \cdot P$

- services expose their protocols + *generic* termination handlers
- services can be deployed dynamically, shut down and updated
- services can handle multiple requests separately

Service invocations: $\overline{s}_k.P$, $k \cdot P$

- service invocations expose their protocols + *specific* termination handlers
- sequential composition via pipelining (á la Orc)

Session termination: $r \triangleright_k P$, close , $\blacktriangleright P$, $\dagger(k)$

- local session termination: autonomous + on partner's request
- the local closure of a session activates partner's handler (if any)
- session termination cancels all locally nested processes (including service definitions) + informs their partners

Termination Handlers

Step 1: Exchanging information about handlers

$\overline{s}_{k_1}.Q|s_{k_2}.P$ can evolve to $(\nu r)(r \rhd_{k_2} Q | r \rhd_{k_1} P)$

Step 2: Closing own session

$r \rhd_k (\text{close} \mid P)$ can evolve to $\dagger(k) | \blacktriangleright P$

Step 3: Structural congruence (see Figure 7) + Propagation

$$\blacktriangleright r \rhd_k P \quad \equiv \quad \blacktriangleright r \rhd_k \blacktriangleright P \qquad \blacktriangleright (P > Q) \quad \equiv \quad (\blacktriangleright P) > Q$$

$$\blacktriangleright r \rhd_k P \quad \xrightarrow{\tau} \quad \blacktriangleright P | \dagger(k)$$

Step 4: Inform handlers

$$k \cdot P \xrightarrow{k} P \qquad \dagger(k) \xrightarrow{\overline{k}} \mathbf{0} \qquad \frac{P \xrightarrow{k} P' \quad Q \xrightarrow{\overline{k}} Q'}{P|Q \xrightarrow{\tau} P'|Q'}$$

Default closing policy

$(\nu k_1)\overline{s}_{k_1}.(P_1 | k_1 \cdot \text{close}\,)$ and $(\nu k_2)s_{k_2}.(P_2 | k_2 \cdot \text{close}\,)$

Termination Handlers

Step 1: Exchanging information about handlers

$$\overline{s}_{k_1}.Q|s_{k_2}.P \quad \text{can evolve to} \quad (\nu r)(r \rhd_{k_2} Q|r \rhd_{k_1} P)$$

Step 2: Closing own session

$$r \rhd_k (\mathsf{close} \mid P) \quad \text{can evolve to} \quad \dagger(k)| \blacktriangleright P$$

Step 3: Structural congruence (see Figure 7) + Propagation

$$\blacktriangleright r \rhd_k P \quad \equiv \quad \blacktriangleright r \rhd_k \blacktriangleright P \qquad \blacktriangleright (P > Q) \quad \equiv \quad (\blacktriangleright P) > Q$$

$$\blacktriangleright r \rhd_k P \quad \xrightarrow{\tau} \quad \blacktriangleright P|\dagger(k)$$

Step 4: Inform handlers

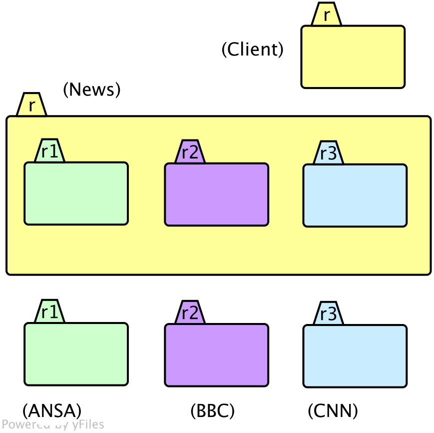
$$k \cdot P \xrightarrow{k} P \qquad \dagger(k) \xrightarrow{\overline{k}} \mathbf{0} \qquad \frac{P \xrightarrow{k} P' \quad Q \xrightarrow{\overline{k}} Q'}{P|Q \xrightarrow{\tau} P'|Q'}$$

Default closing policy

$$(\nu k_1)\overline{s}_{k_1}.(P_1|k_1 \cdot \mathsf{close}) \text{ and } (\nu k_2)s_{k_2}.(P_2|k_2 \cdot \mathsf{close})$$

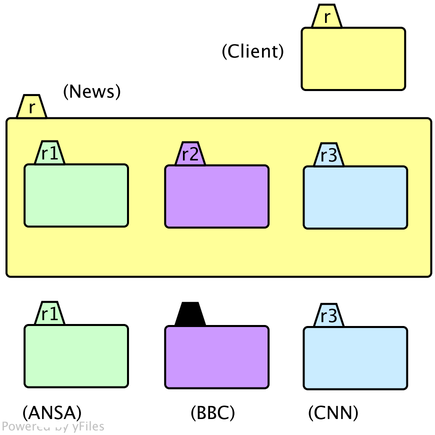
A Last Example: All Sides are Active

$$
\begin{array}{llllll}
News & \triangleq & !(\nu k)\texttt{collect}_k. & (\quad k \cdot \mathsf{close} & | & (\nu k_1)\overline{\texttt{ANSA}}_{k_1}.(!(?x)\langle x\rangle^{\uparrow} \quad | \quad k_1 \cdot (\mathsf{close}\,|\,\dagger(k))) \\
 & & & & | & (\nu k_2)\overline{\texttt{BBC}}_{k_2}.(!(?x)\langle x\rangle^{\uparrow} \quad | \quad k_2 \cdot (\mathsf{close}\,|\,\dagger(k))) \\
 & & & & | & (\nu k_3)\overline{\texttt{CNN}}_{k_3}.(!(?x)\langle x\rangle^{\uparrow} \quad | \quad k_3 \cdot (\mathsf{close}\,|\,\dagger(k)))\ )
\end{array}
$$



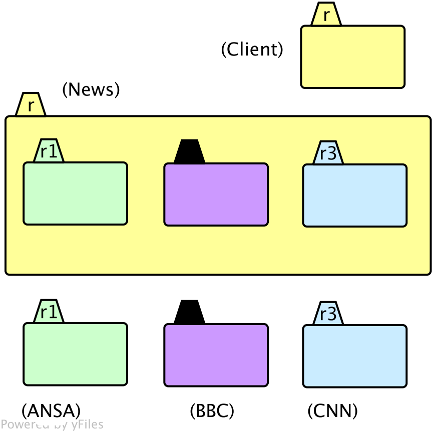
A Last Example: BBC-side Terminates

$$
\begin{array}{llllll}
\textit{News} & \triangleq & !(\nu k)\texttt{collect}_k. & (\quad k \cdot \mathsf{close} & | & (\nu k_1)\overline{\texttt{ANSA}}_{k_1}.(!(?x)\langle x\rangle^{\uparrow} \quad | \quad k_1 \cdot (\mathsf{close}\,|\,\dagger(k))) \\
& & & & | & (\nu k_2)\overline{\texttt{BBC}}_{k_2}.(!(?x)\langle x\rangle^{\uparrow} \quad | \quad k_2 \cdot (\mathsf{close}\,|\,\dagger(k))) \\
& & & & | & (\nu k_3)\overline{\texttt{CNN}}_{k_3}.(!(?x)\langle x\rangle^{\uparrow} \quad | \quad k_3 \cdot (\mathsf{close}\,|\,\dagger(k)))\ )
\end{array}
$$



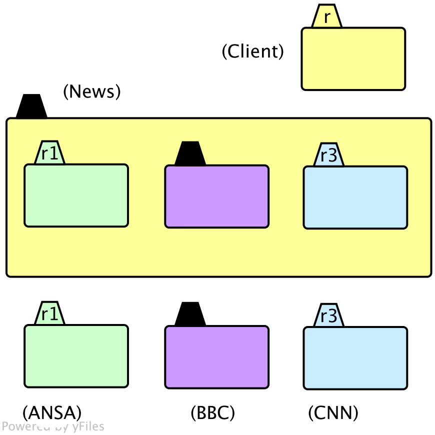
A Last Example: BBC-partner-side Terminates

$$
\begin{array}{llllllll}
\mathit{News} & \triangleq & !(\nu k)\mathtt{collect}_k. & (& k \cdot \mathsf{close} & | & (\nu k_1)\overline{\mathtt{ANSA}}_{k_1}.(!(?x)\langle x\rangle^{\uparrow} & | & k_1 \cdot (\mathsf{close} \,|\, \dagger(k))) \\
 & & & & & | & (\nu k_2)\overline{\mathtt{BBC}}_{k_2}.(!(?x)\langle x\rangle^{\uparrow} & | & k_2 \cdot (\mathsf{close} \,|\, \dagger(k))) \\
 & & & & & | & (\nu k_3)\overline{\mathtt{CNN}}_{k_3}.(!(?x)\langle x\rangle^{\uparrow} & | & k_3 \cdot (\mathsf{close} \,|\, \dagger(k)))\ )
\end{array}
$$



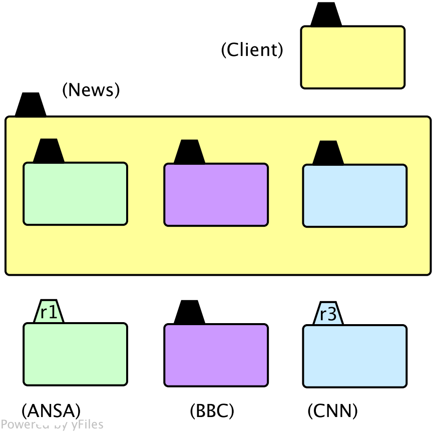
A Last Example: News-side is Triggered to Terminate

$$
\begin{array}{llllllll}
News & \triangleq & !(\nu k)\mathtt{collect}_k. & (& k \cdot \mathsf{close} & | & (\nu k_1)\overline{\mathtt{ANSA}}_{k_1}.(!(?x)\langle x\rangle^{\uparrow} & | & k_1 \cdot (\mathsf{close} \,|\, \dagger(k))) \\
 & & & & & | & (\nu k_2)\overline{\mathtt{BBC}}_{k_2}.(!(?x)\langle x\rangle^{\uparrow} & | & k_2 \cdot (\mathsf{close} \,|\, \dagger(k))) \\
 & & & & & | & (\nu k_3)\overline{\mathtt{CNN}}_{k_3}.(!(?x)\langle x\rangle^{\uparrow} & | & k_3 \cdot (\mathsf{close} \,|\, \dagger(k)))\ )
\end{array}
$$



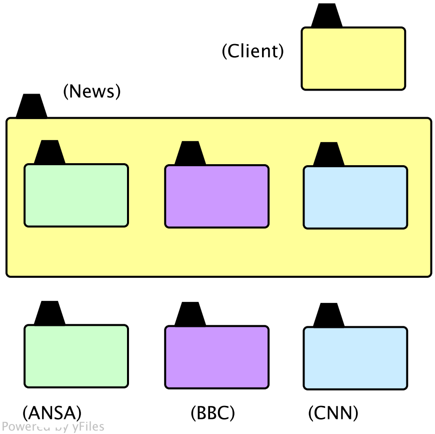
A Last Example: Client-sides and Nested-sides Terminate

$$
\begin{array}{lllllll}
\mathit{News} & \triangleq & !(\nu k)\mathtt{collect}_k. & (\; k \cdot \mathsf{close} & | & (\nu k_1)\overline{\mathtt{ANSA}}_{k_1}.(!(?x)\langle x\rangle^{\uparrow} & | \; k_1 \cdot (\mathsf{close} \,|\, \dagger(k))) \\
 & & & & | & (\nu k_2)\overline{\mathtt{BBC}}_{k_2}.(!(?x)\langle x\rangle^{\uparrow} & | \; k_2 \cdot (\mathsf{close} \,|\, \dagger(k))) \\
 & & & & | & (\nu k_3)\overline{\mathtt{CNN}}_{k_3}.(!(?x)\langle x\rangle^{\uparrow} & | \; k_3 \cdot (\mathsf{close} \,|\, \dagger(k))) \;)
\end{array}
$$



A Last Example: ANSA/CNN-sides Terminate

$$
\begin{array}{llllllll}
News & \triangleq & !(\nu k)\texttt{collect}_k. & (& k \cdot \text{close} & | & (\nu k_1)\overline{\texttt{ANSA}}_{k_1}.(!(?x)\langle x\rangle^{\uparrow} & | & k_1 \cdot (\text{close} \,|\, \dagger(k))) \\
 & & & & & | & (\nu k_2)\overline{\texttt{BBC}}_{k_2}.(!(?x)\langle x\rangle^{\uparrow} & | & k_2 \cdot (\text{close} \,|\, \dagger(k))) \\
 & & & & & | & (\nu k_3)\overline{\texttt{CNN}}_{k_3}.(!(?x)\langle x\rangle^{\uparrow} & | & k_3 \cdot (\text{close} \,|\, \dagger(k)))\ )
\end{array}
$$



Outline

Conclusion and Related Work

CaSPiS

- Original mix of several ingredients
- Flexible and expressive
- Only proposal, up to our knowledge, able to guarantee a disciplined termination of nested sessions.

Related work

- Prototype implementation (as seen in Michele Loreti's talk)
- Type systems available (UGO65, AMAST 2008)
- Type inference is possible (see Leonardo Mezzina's talk)