Calculi for Service-Oriented Computing

Dedicated to NADIA BUSI

Rocco De Nicola

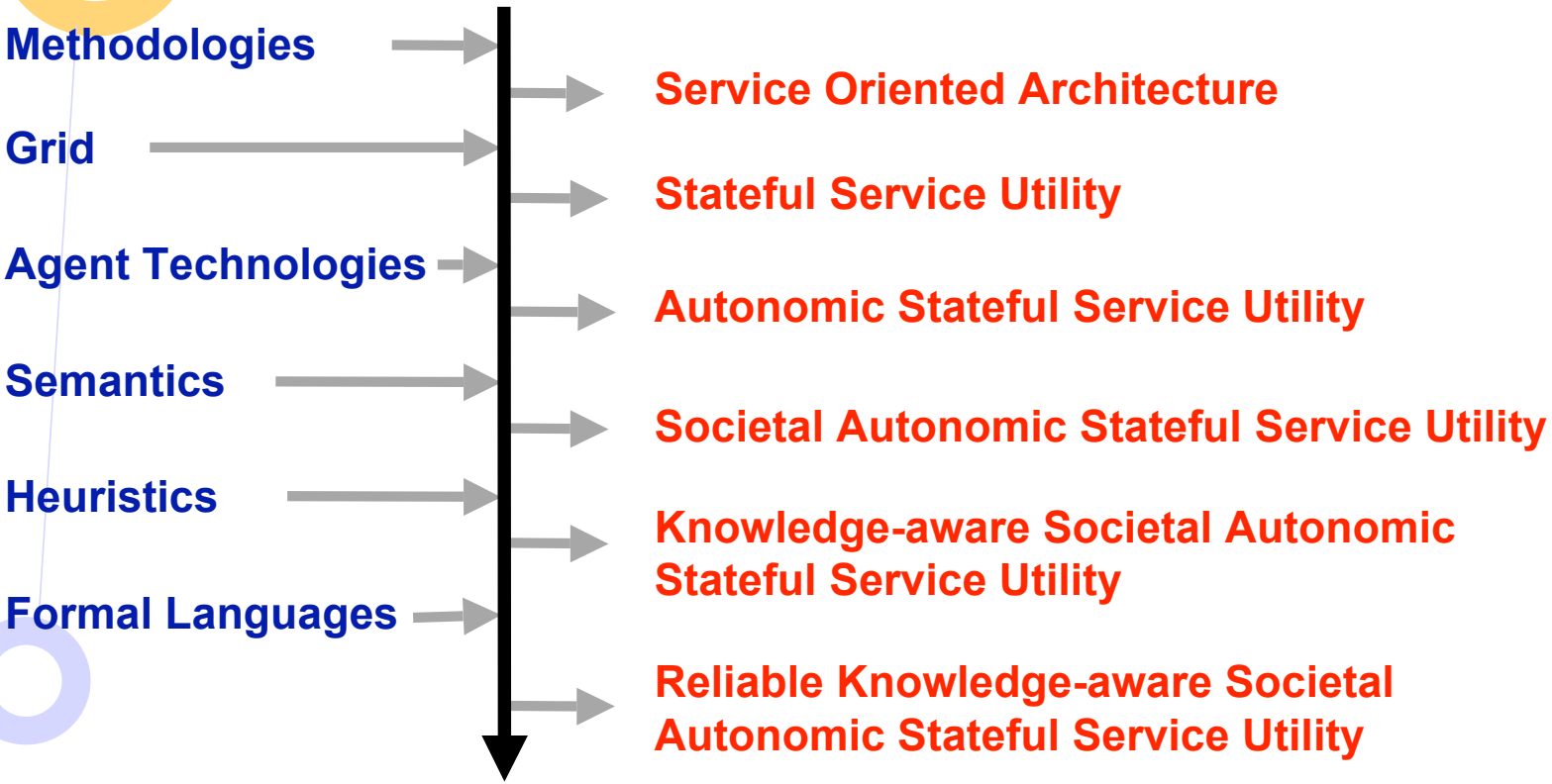
Dipartimento di Sistemi e Informatica
Università di Firenze

Globan Summar School
Warsaw - September 2008

Outline

The services (r)evolution

Internet, Web and Web Services



Service Oriented Knowledge Utility

Service Oriented Computing

Features

Service-oriented computing is an emerging paradigm where services are understood as autonomous and platform-independent computational entities that can be:

- described
- published
- discovered
- dynamically assembled

for developing massively distributed, interoperable, evolvable system.

e-Expectations

Large companies put many efforts in promoting service delivery on a variety of computing platforms. We can anticipate a plethora of new services for . . . e-government, e-business, e-science, . . . and other areas of Information Society.

A crucial point

Industrial consortia are developing orchestration and choreography languages, aiming at setting standards for Web Services and XML-centric technologies, but they lack semantic foundations.

SENSORIA

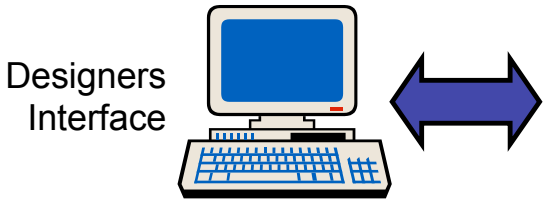
Software Engineering for Service-Oriented Computing

IST-FET Integrated Project funded by the EU
within in the GC2 Initiative.

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

`http://www.sensoria-ist.eu`

A Typical Scenario for SENSORIA Service Design



SENSORIA Development

integrates

practical SW Engineering

with

math. foundations

UML for GC Services
(Model-driven Development)

Automatic connections

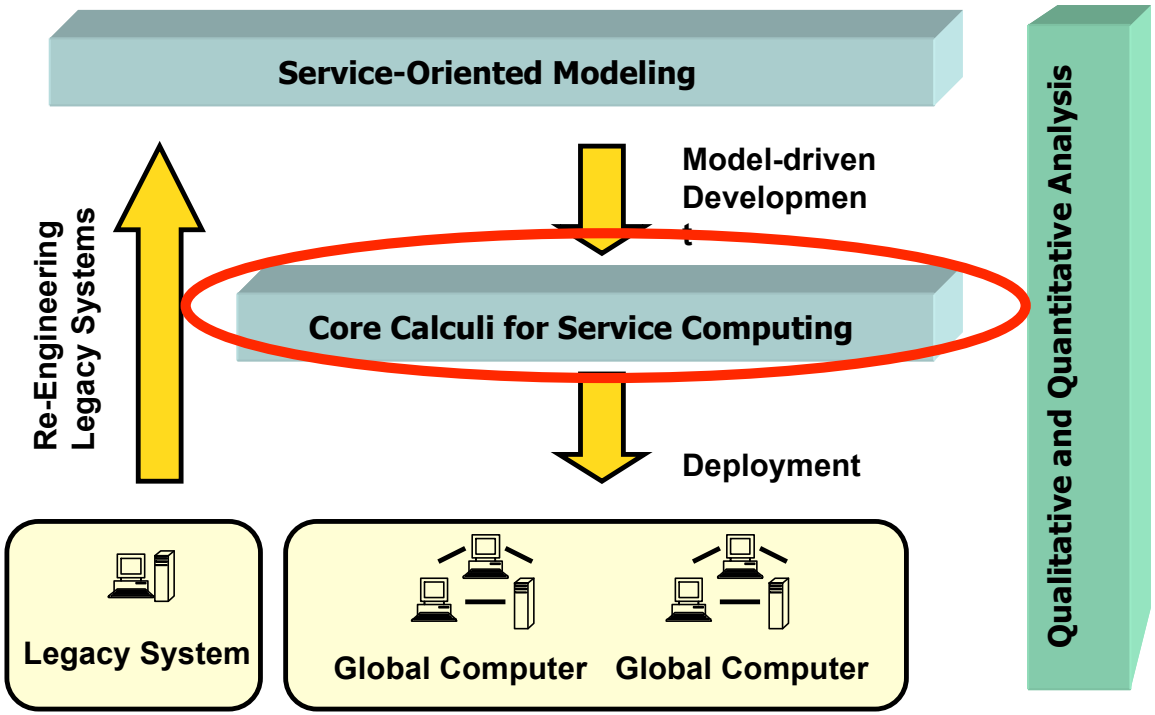
Hidden from the designer

Mathematical Models
(Primitives and languages for GC -services)

Integration
Simulation/verification

Quantitative and Qualitative Properties
(Performance, reliability, fault-tolerance, security, trust, mobility, …)

SENSORIA Detailed Approach



Process Calculi and SENSORIA

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

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

Core Calculi for SOC

Aims

To develop calculi for service specifications and analysis that:

- comply with a service-oriented approach to business modelling;
- allow for modular description of services;
- support dynamic, ad-hoc, "just-in-time" composition.

Assessment of Achievements

The quality of the proposals is being evaluated by means of a number of case studies, while considering crucial aspects like

- expressive power relatively to service contracts, service discovery, service composition, ...
- analytical power relatively to functional and non functional properties (type systems, temporal logics, model checking, stochastic variants, ...)

Basic Ingredients of Core Calculi

Caveat

The outcome of the search for a calculus for SOC will not necessarily be a single calculus. We have competing paradigms related to:

- chosen level of abstraction
- client-service interaction/coordination mechanisms
- primitives for orchestration

Stepping Stones

Most of the proposed calculi are heavily based on process algebras but enhanced with:

- primitives for manipulating semi-structured data (e.g. patt. match.)
- operators for composing (possibly unreliable) services
- techniques for query and discovery of services.
- constructors to properly model interaction

Interaction: A key ingredient of services

Continued Interaction

Service Oriented Programming relies on a lasting interaction between clients and servers and on the exchange of information that influences the progress of both parties.

Stateless protocols

Basic web services are based on stateless interaction protocols (e.g. http) but it is necessary to keep information about partners identities and guarantee appropriate flow of information between the right peers.

An example of stateful service

A server that accepts requests (e.g. seats bookings) from different clients on the same ports but keeps into account the different identities of the clients.

Session: A key notion of stateful services

In SOC an important notion is that of session considered as a logical unit of continued interaction and cooperation between partners.

Modelling Session: The Old Way

- Sessions are simply explicit sequence of communication (possibly controlled via session types)
- Information is shared to uniquely identify partners and keep track of the flow of data and of the protocol progress.

Modelling Session: The New Way

- Correlation Sets a la BPEL
- Explicit Primitives for sessioning

Correlations vs Sessions

Correlation Based Calculi

The link between partners (caller and callee) are determined by correlation sets. An instance of a correlation set contains some correlation values (or variables) and only messages with the right correlation values (or variable instantiations) are received.

Session Based Calculi

A session corresponds to a private channel that is instantiated when calling a service: It binds caller and callee and is used for their communication.

A few calculi

Pre-existing SENSORIA

First, a number of existing calculi has been used:

- CCS, $\pi-$calculus, Join, Ambient, KLAIM, ...

Developed within SENSORIA

Then, a suite of calculi has been proposed, each of them aiming at capturing specific issues of SOC:

- SOCK, COWS, SCC, SC, λ^{req}

Assessment of Calculi

It is important to assess relative merits and expressiveness of the calculi and to look for unifying solutions.

SOCK: a calculus for Service Oriented Computing

Main Aims

- SOCK is a formal calculus which aims at characterizing the basic features of Service Oriented Computing and takes its inspiration from WS-BPEL, a 'de facto' standard for Web Service technology
- C. Guidi, R. Lucchi, R. Gorrieri, N. Busi, G. Zavattaro 4th Int. Conf. on Service-Oriented Computing, LNCS 4294, Springer, 2006

Basic Structure

A three layered structure inspired by the Web services protocol stack.

- The service behaviour calculus provides the primitives for services programming;
- The service engine calculus provides the mechanisms for describing service engines;
- The service system calculus provides the mechanisms for composing/deployng service engines into a system.

COWS: a Calculus for Orchestration of Web Services

Main Aims

- COWS exploits WS-BPEL to drive the design of a foundational calculus to reason on specified services.
- A. Lapadula, R. Pugliese, F. Tiezzi, 16th European Symposium on Programming, LNCS 4421, Springer, 2007

Basic Structure

- Threads of a service can share the store and sessions can be modeled by means of correlation sets.
- Some WS-BPEL constructs, e.g. fault and compensation handlers and flow graphs, do not have direct counterparts; they can be encoded by exploiting COWS operators.
- Borrows ideas from other calculi: asynch. comm., pattern matching, polyadic synchronization, localized input ($L\pi$), delimited kill & protection ($StAC_i$)

SCC: a Service-Centered Calculus

Main Aims

- A a small set of primitives for programming and orchestrating services. A concept of *session* for client-server interaction.
- M. Boreale, R. Bruni, L. Caires, R. De Nicola, I. Lanese, M. Loreti, F. Martins, U. Montanari, A. Ravara, D. Sangiorgi, V. Vasconcelos, G. Zavattaro, 3rd Int. Workshop on Web Services and Formal Methods, LNCS 4184, pages 38-57, Springer, 2006.

Basic Structure

- Looks for a small set of primitives that might serve as a basis for formalising and programming service oriented applications over global computers
- integrates complementary aspects from π-calculus (naming, hence sessions), Orc (pipelining of activities), webπ, cJoin, Sagas (implicit transactions and compensations)

SC: Signal Calculus

Main Aims

- Event-Based Service Coordination with the goal of providing a semantic based framework to implement higher level languages e.g. WS-CDL, BPEL, SAGA, . . . , and a common programming model for service coordination (choreography)
- G. Ferrari, R. Guanciale, D. Strollo, 4th Int. Conference on Service-Oriented Computing, LNCS 4294, Springer, 2006

Basic Structure

- A process calculus with asynchronous communication based on the Event Notification paradigm
- Uses event topics to describe coordination policies
- Permits dynamic interface publication and modification of service connections.

λ^{req} (or CCC): A Calculus for Call by Contract

Main Aims

- Calculus based on call-by-contract service selection that guarantees security-awareness from the design phase and supports verification techniques for planning sound compositions
- M. Bartoletti, P. Degano, G.L. Ferrari, Journal of Computer Security, 2007.

Basic Structure

- Selects and configures services, to guarantee that their composition enjoys some desirable properties
- Takes into account non-functional aspects: security and contracts
- Analyzes service infrastructure to see which features are needed to guarantee semantic properties on service behaviour

Assessing the Calculi

Different abstraction levels and different aims

SC aims at providing a basic framework (a middleware?) for implementing languages for SOC;

SOCK is an abstraction of BPEL approach but sufficiently close to it to mimic its development process;

COWS starts from BPEL but abstract more from it to get very close to a classical process description language;

SCC starts from the abstract notion of session to develop a calculus along the lines of π-calculus;

λ^{req} is specifically designed for call-by-contract service selection and the advocated paradigm could be used to extend any of the previous formalisms.

Three Variants of SCC

PSCC (aka CaSPiS): Dataflow oriented

Boreale, Bruni, De Nicola, Loreti:
A pipelining operator (*à la* ORC) is introduced to model the passage of information between sessions; a return command is used by sessions for passing values to the environment.

SSCC: Stream oriented

Lanese, Martins, Ravara, Vasconcelos:
Relies on explicit streams; primitives for inserting/retrieving data into/from streams are used both for inter-session communication and for communicating with the environment.

CSCC: Message passing oriented

Caires, Vieira:
Three distinct set of primitives are used for inter-process, for inter-session comm. and for communicating with the environment.

CaSPiS in a nutshell...

Service definitions and invocations...

...are rendered respectively as $s.P$ and $\overline{s}.Q$:

- s is a service name
- P and Q implement the service and the client *protocols*

$$\texttt{news}.\langle \text{“}\textit{news item}\text{”} \rangle P \mid \overline{\texttt{news}}.(?x)\langle x \rangle^{\uparrow} Q$$

$$\downarrow$$

$$(\nu r)\left(r \rhd \langle \text{“}\textit{news item}\text{”} \rangle P \mid r \rhd (?x)\langle x \rangle^{\uparrow} Q\right)$$

CaSPiS in a nutshell...

Abstractions and concretions...

Processes at the two sides of a session can interact with each other by means of:

- *concretions*: $\langle V \rangle P$ sends value V over a session
- *abstractions*: $(F)P$ retrieves a value matching pattern F.

$$(\nu r)\left(r \triangleright \langle \text{“}\textit{news item}\text{”} \rangle P \mid r \triangleright (?x)\langle x \rangle^{\uparrow} Q\right)$$
$$\downarrow$$
$$(\nu r)\left(r \triangleright P \mid r \triangleright \langle \text{“}\textit{news item}\text{”} \rangle^{\uparrow} Q\right)$$

Return...

Values can be returned outside a session to the enclosing environment using the return operator, $\langle \cdot \rangle^{\uparrow}$.

CaSPiS in a nutshell...

Pipeline...

Values returned by a session can be used to start new activities. This is achieved using the *pipeline* operator:

$$P > Q\,.$$

A *new* instance of process Q is activated each time P emits a value that Q can consume.

$$r \triangleright \langle \text{“}\textit{news item}\text{”} \rangle^{\uparrow} P' > (?z)\overline{\texttt{emailMe}}.\langle z \rangle \mathbf{0}$$

$$\downarrow$$

$$r \triangleright P' > (?z)\overline{\texttt{emailMe}}.\langle z \rangle \mathbf{0} \mid \overline{\texttt{emailMe}}.\langle \text{“}\textit{news item}\text{”} \rangle \mathbf{0}$$

A Simple Example

News Collector:

News services, say CNN and BBC, periodically send news to registered users:

$$\texttt{CNN}.\mathbf{rec}\ X.\langle news_{cnn}\rangle X \qquad \texttt{BBC}.\mathbf{rec}\ X.\langle news_{bbc}\rangle X$$

The following process models a service that automatically sends us an email with the news provided by CNN and BBC:

$$\left(\begin{array}{l} \overline{\texttt{CNN}}.\mathbf{rec}\ X.(?x)\langle x\rangle^{\uparrow} X \\ \mid \overline{\texttt{BBC}}.\mathbf{rec}\ Y.(?y)\langle y\rangle^{\uparrow} Y \end{array}\right) > (?m)\overline{emailMe}.\langle m\rangle \mathbf{0}$$

CaSPiS Syntax (Close-free fragment)

P, Q	$::=$	$\sum_{i \in I} \pi_i P_i$	Guarded Sum	π	$::=$	(F) Abstraction
	$\mid$	$s.P$	Service Definition	$\mid$		$\langle V \rangle$ Concretion
	$\mid$	$\overline{s}.P$	Service Invocation	$\mid$		$\langle V \rangle^{\uparrow}$ Return
	$\mid$	$r \triangleright P$	Session			
	$\mid$	$P > Q$	Pipeline			
	$\mid$	$P \mid Q$	Parallel Composition			
	$\mid$	$(\nu n)P$	Restriction			
	$\mid$	$!P$	Replication			

V	$::=$	$u \mid f(\tilde{V}) \ \ (f \in \Sigma)$	Value
F	$::=$	$?x \mid u \mid f(\tilde{F}) \ \ (f \in \Sigma)$	Pattern

CaSPiS: Abstractions and Concretions

Concretion

$\langle V \rangle P$ produces a value V and then behaves like P:

$$\langle V \rangle P \xrightarrow{\langle V \rangle} P$$

Abstraction

$(F)P$ waits for a value matching F and then activates process P:

$$\frac{\mathrm{match}(F, V) = \sigma}{(F)P \xrightarrow{(V)} P\sigma}$$

Service Definition and Service Invocation

- $s.P$ identifies a definition for service s with body P.
- $\overline{s}.Q$ invokes service s then continues (waits?) according to protocol Q.
- A service invocation causes the activation of a new session; a fresh name r ($r \notin \mathrm{fn}\, P$) identifies the two sides of the session.

$$s.P \xrightarrow{s(r)} r \triangleright P \qquad \overline{s}.Q \xrightarrow{\overline{s}(r)} r \triangleright Q$$

$$\frac{P \xrightarrow{s(r)} P' \quad Q \xrightarrow{\overline{s}(r)} Q'}{P|Q \xrightarrow{\tau} (\nu r)(P'|Q')}$$

Session

$r \triangleright [.]$ is an abstract channel for Client and Server interaction:

$$\frac{P \xrightarrow{\langle V \rangle} P'}{r \triangleright P \xrightarrow{r:\langle V \rangle} r \triangleright P'} \qquad \frac{P \xrightarrow{(V)} P'}{r \triangleright P \xrightarrow{r:(V)} r \triangleright P'}$$

$$\frac{P \xrightarrow{r:(V)} P' \quad Q \xrightarrow{r:\langle V \rangle} Q'}{P|Q \xrightarrow{\tau} P'|Q'}$$

$$\frac{P \xrightarrow{\lambda} P'}{r \triangleright P \xrightarrow{\lambda} r \triangleright P'} \quad \lambda \neq (V), \langle V \rangle, \langle V \rangle^{\uparrow}$$

CaSPiS: Inter-session interactions

Return:

$\langle V \rangle^{\uparrow} P$ can be used to return a value to the enclosing environment:

$$\langle V \rangle^{\uparrow} P \xrightarrow{\uparrow V} P \qquad \frac{P \xrightarrow{\uparrow V} P'}{r \rhd P \xrightarrow{\langle V \rangle} r \rhd P'}$$

Pipeline

Different activities can be composed by using pipeline $P > Q$. Each value produced by P activates a new instance Q' of Q:

$$\frac{P \xrightarrow{\langle V \rangle} P' \quad Q \xrightarrow{(V)} Q'}{P > Q \xrightarrow{\tau} (P' > Q) | Q'} \qquad \frac{P \xrightarrow{\lambda} P' \quad \lambda \neq \langle V \rangle}{P > Q \xrightarrow{\lambda} P' > Q}$$

CaSPiS at Work: Derivable Operators

Invocation Patterns

Invocation patterns are shorthands for common (standard?) ways of invoking services and waiting for their results:

- One Way: $\overline{s}\langle V\rangle \stackrel{\triangle}{=} \overline{s}.\langle V\rangle$
- Request Response: $\overline{s}(V) \stackrel{\triangle}{=} \overline{s}.\langle V\rangle(?x)\langle x\rangle^{\uparrow}$
- Get All Responses: $\overline{s}(!) \stackrel{\triangle}{=} \overline{s}.!(?x)\langle x\rangle^{\uparrow}$
- Simple Pipe: $P > \overline{s} \stackrel{\triangle}{=} P > (?x)\overline{s}(x)$

Example:

$$\langle 5\rangle > \overline{\texttt{succ}} > \overline{\texttt{succ}}$$

CaSPiS at Work: Derivable Operators

Selection

select $F_1, \ldots, F_n$ **from** P **in** Q collects the first n values emitted by a set of parallel processes satisfying a given sequence of patterns:

$$\textbf{select } F_1, \ldots, F_n \textbf{ from } P \stackrel{\triangle}{=} (\nu s)\left(s.(F_1)\ldots(F_n)\langle \hat{F}_1, \ldots, \hat{F}_n\rangle^{\uparrow} \mid \overline{s}.P\right)$$

where for each pattern F_i, $\hat{F}_i$ denotes the value V_i obtained from F_i by replacing each $?x$ with x.

Example:

$$\textbf{select } ?x, ?y \textbf{ from } \left(\overline{\texttt{ANSA}}(!) \mid \overline{\texttt{BBC}}(!) \mid \overline{\texttt{CNN}}(!)\right) \textbf{ in } \overline{\texttt{emailMe}}\langle x, y\rangle$$

CaSPiS at Work: Derivable Operators

Waiting

wait $F_1, \ldots, F_n$ **from** $P_1, \ldots, P_n$ **in** Q waits for tuple $\langle V_1, \ldots, V_n \rangle$, where V_i is the first value emitted by P_i and matching F_i, for $i = 1, ..., n$, then activates process Q.

$$
\begin{array}{l}
\textbf{wait } F_1, \ldots, F_n \textbf{ from } P_1, \ldots, P_n \triangleq \\
\quad (\nu s)(\\
\quad\quad \overline{s}.(\ \textbf{select } F_1 \textbf{ from } P_1 \textbf{ in } \langle t_1(\hat{F}_1) \rangle \\
\quad\quad\quad \mid \ \ldots \\
\quad\quad\quad \mid \textbf{select } F_n \textbf{ from } P_n \textbf{ in } \langle t_n(\hat{F}_n) \rangle\ ) \\
\quad\quad \mid s.(t_1(F_1)).\ldots.(t_n(F_n))\langle \hat{F}_1, \ldots, \hat{F}_n \rangle^{\uparrow} \\
\quad\quad\quad)
\end{array}
$$

Example:

wait $?x, ?y, ?z$ **from** $\overline{\texttt{ANSA}}(!)$, $\overline{\texttt{BBC}}(!)$, $\overline{\texttt{CNN}}(!)$ **in** $\overline{\texttt{emailMe}}\langle x, y, z \rangle$

CaSPiS at Work: Iterating

Iteration

process iteration P is programmed so that each time process P emits a value it is restarted. Assuming a default value $\bullet$, iteration can be defined as follows:

$$*P \stackrel{\triangle}{=} (\nu s)\big(\ \overline{s}.\mathbf{0} \mid (!s.\langle\bullet\rangle^{\uparrow}) > (\bullet)\mathbf{select}\ ?x\ \mathbf{from}\ P\ \mathbf{in}\ \langle x\rangle\overline{s}.\mathbf{0}\ \big)$$

Example:

Iterator can be used for implementing a process that continuously sends emails with news produced by `BBC` and `CNN`:

$$*(\ \mathbf{wait}\ ?x, ?y\ \mathbf{from}\ \overline{\texttt{BBC}}, \overline{\texttt{CNN}}\ ) > (?x, ?y)\overline{\texttt{emailMe}}\langle x, y\rangle$$

CaSPiS at Work: A travel Service

A travel agent offers its customers the ability to book packages consisting of services offered by different providers.

- Three kinds of booking are available:
  - Flight only,
  - Hotel only,
  - Both flight and hotel.

A customer contacts the travel agent service and then provides the appropriate information about the planned travel (origin and destination, departure and return dates,...).

When a request is received, the service contacts appropriate services (for instance `Lufth` and `Alit` for flights, and `HInn` and `BWest` for hotels) and then compares their offers to select the most convenient

CaSPiS at Work: Example

$$
\begin{aligned}
&!\mathtt{ta}.\\
&\quad (\mathit{fly}(?x)).\\
&\qquad \textbf{wait } ?y, ?z \textbf{ from } \overline{\mathtt{Lufth}}\langle x\rangle, \overline{\mathtt{Alit}}\langle x\rangle \textbf{ in } \overline{\mathtt{compare}}\langle (y,z)\rangle\\
&\quad + (\mathit{hotel}(?x)).\\
&\qquad \textbf{wait } ?y, ?z \textbf{ from } \overline{\mathtt{BWest}}\langle x\rangle, \overline{\mathtt{HInn}}\langle x\rangle \textbf{ in } \overline{\mathtt{compare}}\langle (y,z)\rangle\\
&\quad + (\mathit{fly}\&\mathit{hotel}(?x)).\\
&\qquad \textbf{wait } ?y, ?z \textbf{ from } \overline{\mathtt{ta}}\langle \mathit{fly}(x)\rangle, \overline{\mathtt{ta}}\langle \mathit{hotel}(x)\rangle \textbf{ in } \langle y,z\rangle
\end{aligned}
$$

Quantitative Analysis of Systems

Besides qualitative aspects of service oriented systems it is also important that phenomena related to performance and dependability are addressed, e.g., to deal with issues related to quality of service.

These aspects are particularly relevant for systems interacting over wide area networks where

- components failures are likely;
- congestions may cause unpredictable delays.

We have introduced a versatile technique for the definition of structured operational semantics of stochastic extensions of process calculi and have used it to provide

- A Markovian extension for CaSPiS: MarCaSPiS

MarCaSPiS: Markovian CaSPiS

MarCaSPiS is a Markovian extension of CaSPiS where:

- *output activities* are enriched with *rates* characterizing random variables with exponential distributions, modeling their duration;
- *input activities* are equipped with *weights* characterizing the relative selection probability

Continuous Time Markov Chains(CTMS) for MarCaSPiS specifications are obtained by considering only internal actions and service activations

Existing tools can be used for supporting quantitative analysis of MarCaSPiS specification.

Stochastic process calculi...

- Actions execution takes time
- Execution times is described by means of Random Variables
- Random Variables are assumed to be Exponentially Distributed
- Random Variables are fully characterised by their rate

If a random variable X is *exponentially distributed* with *parameter* $\lambda \in \mathbb{R}_{>0}$:

- $\mathbb{P}\{X \leq d\} = 1 - e^{-\lambda \cdot d}$, for $d \geq 0$
- The average duration of X is $\frac{1}{\lambda}$;
- $\mathbb{P}\{X \leq t + d \mid X > t\} = \mathbb{P}\{X \leq d\}$ (*Memory-less*)

Stochastic process calculi...

A Continuous Time Markov Chain (CTMC) is associated to each term of the process algebras. This will be used for defining the stochastic behaviour of processes.

We have to...

- compute *synchronizations rate*
- take into account transition multiplicity, for determining correct execution rate

Stochastic Process Calculi:

$$\alpha^{\lambda}.P + \alpha^{\lambda}.P \quad \neq \quad \alpha^{2\lambda}.P$$

$$\textbf{rec}\ X.\alpha^{\lambda}.X \mid \textbf{rec}\ X.\alpha^{\lambda}.X \quad \neq \quad \textbf{rec}\ X.\alpha^{2\lambda}.X$$

Which rate for synchronisations?

1. Natural Synchronization:

The duration of the interaction is distributed as the maximum of the individual distributions:

- once that occurs, each agent proceeds indipendently
- the first to finish will wait for the other agent to also finish before the interaction is terminated

Interaction starts when both agents are ready!

Drawback: The maximum of two exponential distributions **is not** an exponential distribution!

Which rate for synchronisations?

2. Patient Synchronization (PEPA): $\min\{\lambda_1, \lambda_2\}$

- The modelling of agents captures the rate at which actions can be completed;
- It is assumed that both agents have the rate of the slower one;
- This approach is useful for modelling shared tasks.

3. Cooperative Synchronization (TIPP): $\lambda_1 \cdot \lambda_2$

- The presence of another agent may influence how an agent completes an action;
- Agents increase their capacity by working together.

Which rate for synchronisations?

4. Provider Synchronization: (λ_1)

- A provider offers a resource a *service* to requiring clients.
  - one partner is *active*
  - the other partner is *passive*
  - the used rate is the one of the actor.
- There is an asymmetry in the interaction since the rate is completely dominated by one of the participants

5. Sequentialized Synchronization: $\left(\frac{1}{\lambda_1} + \frac{1}{\lambda_2}\right)^{-1}$

- The interaction between the agents is viewed as a series of smaller interactions in which only one agent is active at a time

MarCaSPiS Syntax

$$
\begin{array}{rcll}
P, Q & ::= & \sum_{i \in I} \pi_i P_i & \text{Guarded Sum} \\
& | & \sum_{i \in I} s_i^{\omega_i}.P_i & \text{Service Definitions} \\
& | & \sum_{i \in I} \overline{s_i}^{\lambda_i}.P_i & \text{Service Invocations} \\
& | & r \rhd P & \text{Session} \\
& | & P > Q & \text{Pipeline} \\
& | & P | Q & \text{Parallel Composition} \\
& | & (\nu n)P & \text{Restriction} \\
& | & \textbf{rec}\ X.P & \text{Recursion} \\
& | & X & \text{Process Variable} \\
\pi & ::= & (F)^{\omega} \mid \langle V \rangle^{\lambda} \mid \langle V \rangle^{\uparrow^{\lambda}} & \\
V & ::= & u \mid f(\tilde{V}) \quad (f \in \Sigma) & \\
F & ::= & ?x \mid u \mid f(\tilde{F}) \quad (f \in \Sigma) &
\end{array}
$$

MarCaSPiS: Markovian CaSPiS

Operational semantics of MarCaSPiS

- makes use of *total weights* for determining synchronisation rates (inspired by Hillston's apparent rates)
- relies on a novel approach for handling transition multiplicity

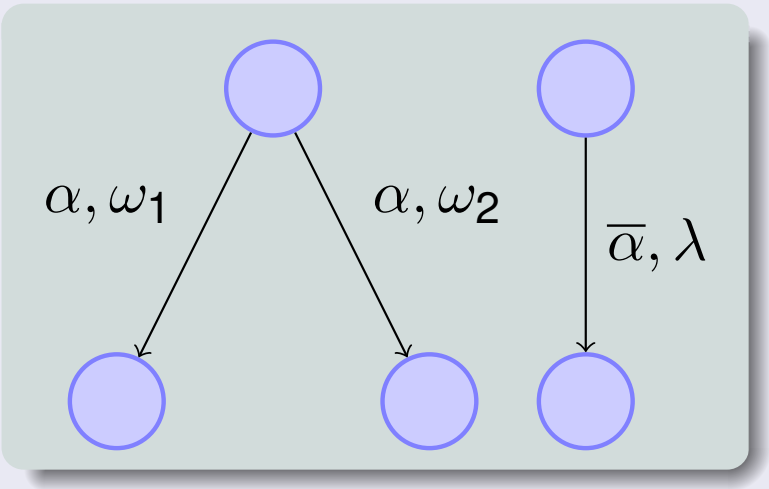
The synchronisation pattern used in MarCaSPiS guarantees associativity of (CCS-like) parallel composition

- this property is lost if a more direct adaptation of Hillston's approach is used in presence of binary synchronization;

- *Total rates* have a *global* meaning, that depends also on the environment where the actiion is performed. Action rates are "re-calculated" on synchronization.
- *Apparent rates* are specific of the components that performs the actions and do not depend on the context the actions take place.

Transitions rates

- The rate of a transition basically depends on the rate of the triggering *activity*
- The synchronization rate of $\overline{\alpha}$ and α depends on the rate of $\overline{\alpha}$, the weight of *selected* α and on the *total weight* of α.
  - the *total weight* of α is the *sum* of the weights of *all* α-transitions



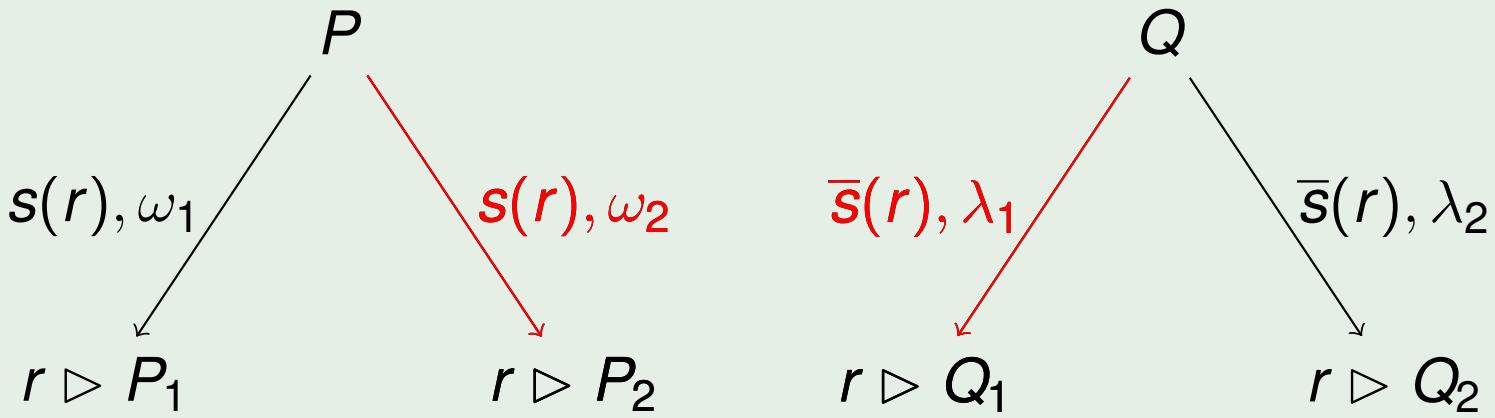
- Two synchronizations can occur with rates:

$$\lambda \cdot \frac{\omega_1}{\omega_1 + \omega_2} \qquad \lambda \cdot \frac{\omega_2}{\omega_1 + \omega_2}$$

- *Total synchronization rate* does not depend on the number of available (input) partners

Computing the rate of a synchronization

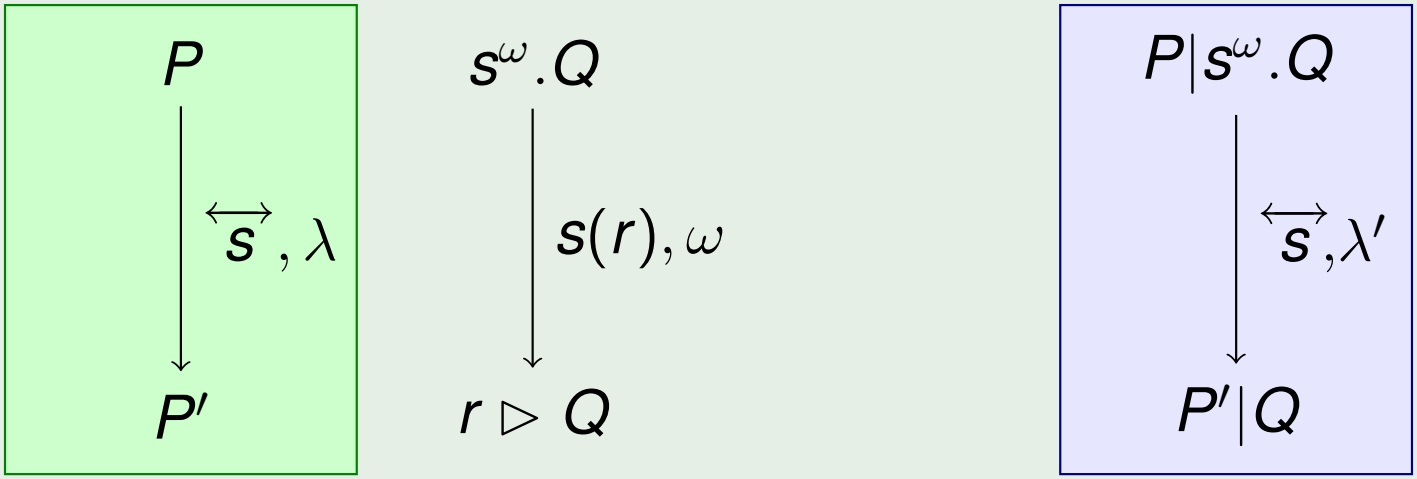
Let $P \stackrel{\triangle}{=} s^{\omega_1}.P_1 + s^{\omega_2}.P_2$ and $Q \stackrel{\triangle}{=} \overline{s}^{\lambda_1}.Q_1 + \overline{s}^{\lambda_2}.Q_2$



Service activation:

$$P|Q \xrightarrow{\overleftrightarrow{s}, \lambda_1 \cdot \frac{\omega_2}{\omega_1+\omega_2}} (\nu r)(r \rhd P_2 | r \rhd Q_1)$$

Computing the rate of a synchronization



where:

$$\lambda' = \lambda \cdot \frac{\overline{\omega}}{\overline{\omega} + \omega}$$

whenever $\overline{\omega}$ indicates the total weight of $s(r)$ in P.

This is crucial to guarantee associativity of parallel composition.

A new approach to stochastic semantics

Stochastic semantics of MarCaSPiS is defined by means of a transition relation $\longrightarrow$ that associates to a process P and a transition label α a function $(\mathscr{P}, \mathscr{Q}, \ldots)$ that maps each process into a non-negative real number.

$P \xrightarrow{\alpha} \mathscr{P}$ means that:

- when $\mathscr{P}(Q) = x$ $(\neq 0)$, Q is reachable from P via the execution of α with rate or weight x
- when $\mathscr{P}(Q) = 0$, Q is not reachable from P via α

We have that if $P \xrightarrow{\alpha} \mathscr{P}$ then

- $\oplus\mathscr{P} = \sum_Q \mathscr{P}(Q)$ represents the total rate/weight of α in P.

MarCaSPiS stochastic semantics

Some basic rule:

$$\frac{r \notin \mathrm{fn}(P)}{s^{\lambda}.P \xrightarrow{s(r)} [r \rhd P \mapsto \lambda]}$$

$$\frac{\alpha \neq s(r) \quad \mathrm{bn}(\alpha) \cap \mathrm{fn}(P) \textit{ is empty}}{s^{\lambda}.P \xrightarrow{\alpha} \emptyset}$$

$$\frac{P \xrightarrow{\alpha} \mathscr{P} \quad Q \xrightarrow{\alpha} \mathscr{Q}}{P + Q \xrightarrow{\alpha} \mathscr{P} + \mathscr{Q}}$$

$$\frac{P \xrightarrow{\alpha} \mathscr{P} \quad Q \xrightarrow{\alpha} \mathscr{Q}}{P|Q \xrightarrow{\alpha} \mathscr{P}|Q + P|\mathscr{Q}}(\alpha \neq \overleftrightarrow{s})$$

where:

- $\emptyset$ denotes the 0 constant function.
- $\mathscr{P} + \mathscr{Q}$ is the function $\mathscr{R}$: $\mathscr{R}(P) = \mathscr{P}(P) + \mathscr{Q}(P)$
- $\mathscr{P}|Q$ is the function $\mathscr{R}'$: $\mathscr{R}'(R) = \mathscr{P}(P)$ if $R = P|Q$, 0 otherwise.

MarCaSPiS stochastic semantics

$$\frac{r \notin \mathrm{fn}(P) \qquad r \notin \mathrm{fn}(P)}{\overline{s}^{\lambda}.P \xrightarrow{\overline{s}(r)} [r \rhd P \mapsto \lambda] \qquad \overline{s}^{\lambda}.P \xrightarrow{\overline{s}(r)} [r \rhd P \mapsto \lambda]}$$

$$\overline{s}^{\lambda}.P + \overline{s}^{\lambda}.P \xrightarrow{\overline{s}(r)} [r \rhd P \mapsto \lambda] + [r \rhd P \mapsto \lambda][r \rhd P \mapsto 2\lambda]$$

$$\frac{\vdots}{\mathbf{rec}\ X.\langle V\rangle^{\lambda}X \xrightarrow{\langle V\rangle} [\mathbf{rec}\ X.\langle V\rangle^{\lambda}X \mapsto \lambda] \qquad \mathbf{rec}\ X.\langle V\rangle^{\lambda}X}$$

$$\mathbf{rec}\ X.\langle V\rangle^{\lambda}X \mid \mathbf{rec}\ X.\langle V\rangle^{\lambda}X \xrightarrow{\langle V\rangle} \begin{array}{l} [\mathbf{rec}\ X.\langle V\rangle^{\lambda}X \mapsto \lambda] \mid \mathbf{rec}\ X.\langle V\rangle^{\lambda}X \quad [\mathbf{rec}\ X.\langle V\rangle^{\lambda}X \mid \\ + \\ \mathbf{rec}\ X.\langle V\rangle^{\lambda}X \mid [\mathbf{rec}\ X.\langle V\rangle^{\lambda}X \mapsto \lambda] \quad [\mathbf{rec}\ X.\langle V\rangle^{\lambda}X \mid \end{array}$$

MarCaSPiS stochastic semantics

A synchronization rule:

$$\frac{P \xrightarrow{\overleftrightarrow{s}} \mathscr{P} \quad P \xrightarrow{s(r)} \mathscr{P}_d \quad P \xrightarrow{\overline{s}(r)} \mathscr{P}_i \quad Q \xrightarrow{\overleftrightarrow{s}} \mathscr{Q} \quad Q \xrightarrow{s(r)} \mathscr{Q}_d \quad Q \xrightarrow{\overline{s}(r)} \mathscr{Q}_i}{P|Q \xrightarrow{\overleftrightarrow{s}} \frac{\mathscr{P}|Q \cdot \oplus\mathscr{P}_d}{\oplus\mathscr{P}_d + \oplus\mathscr{Q}_d} + \frac{P|\mathscr{Q} \cdot \oplus\mathscr{P}_d}{\oplus\mathscr{P}_d + \oplus\mathscr{Q}_d} + \frac{(\nu r)(\mathscr{P}_d|\mathscr{Q}_i)}{\oplus\mathscr{P}_d + \oplus\mathscr{Q}_d} + \frac{(\nu r)(\mathscr{P}_i|\mathscr{Q}_d)}{\oplus\mathscr{P}_d + \oplus\mathscr{Q}_d}}$$

where:

- $\frac{\mathscr{P}\cdot\omega_1}{\omega_2} = \emptyset$ if $\omega_2 = 0$,
- $\frac{\mathscr{P}\cdot\omega_1}{\omega_2} = \mathscr{R}$ if $\omega_2 \neq 0$ and $\mathscr{R}(P) = \frac{\mathscr{P}(P)\cdot\omega_1}{\omega_2}$

Activations of service s in $P|Q$ are determined by considering

- the activations in P, where synchronization rates are updated for considering definitions in Q;
- the activations in Q, where synchronization rates are updated for considering definitions in P;
- interactions between definitions in P with invocations in Q;

A derivation

$$\frac{P \xrightarrow{\overleftrightarrow{s}} \mathscr{P} \quad P \xrightarrow{s(r)} \mathscr{P}_d \quad P \xrightarrow{\overline{s}(r)} \mathscr{P}_i \quad Q \xrightarrow{\overleftrightarrow{s}} \mathscr{Q} \quad Q \xrightarrow{\overline{s}(r)} \mathscr{Q}_i \quad Q \xrightarrow{s(r)} \mathscr{Q}_d}{P|Q \xrightarrow{\overleftrightarrow{s}} \frac{\mathscr{P}|Q \cdot \oplus \mathscr{P}_d}{\oplus \mathscr{P}_d + \oplus \mathscr{Q}_d} + \frac{P|\mathscr{Q} \cdot \oplus \mathscr{P}_d}{\oplus \mathscr{P}_d + \oplus \mathscr{Q}_d} + \frac{(\nu r)(\mathscr{P}_d|\mathscr{Q}_i)}{\oplus \mathscr{P}_d + \oplus \mathscr{Q}_d} + \frac{(\nu r)(\mathscr{P}_i|\mathscr{Q}_d)}{\oplus \mathscr{P}_d + \oplus \mathscr{Q}_d}}$$

$$\frac{\begin{array}{lll} s^5.P \xrightarrow{\overleftrightarrow{s}} \emptyset & s^5.P \xrightarrow{s(r)} [r \triangleright P \mapsto 5] & s^5.P \xrightarrow{\overline{s}(r)} \emptyset \\ \overline{s}^7.Q \xrightarrow{\overleftrightarrow{s}} \emptyset & \overline{s}^7.Q \xrightarrow{\overline{s}(r)} \emptyset & \overline{s}^7.Q \xrightarrow{s(r)} [r \triangleright Q \mapsto 7] \end{array}}{s^5.P|\overline{s}^7.Q \xrightarrow{\overleftrightarrow{s}} \frac{(\nu r)[r \triangleright P \mapsto 5]|[r \triangleright Q \mapsto 7]}{\oplus [r \triangleright P \mapsto 5]} \quad \frac{[(\nu r)(r \triangleright P|r \triangleright Q) \mapsto 35]}{5} \quad [(\nu r)(r \triangleright P|r \triangleright Q$$

$$\frac{P \xrightarrow{\overleftrightarrow{s}} \mathscr{P} \quad P \xrightarrow{s(r)} \mathscr{P}_d \quad P \xrightarrow{\overline{s}(r)} \mathscr{P}_i \quad Q \xrightarrow{\overleftrightarrow{s}} \mathscr{Q} \quad Q \xrightarrow{\overline{s}(r)} \mathscr{Q}_i \quad Q \xrightarrow{s(r)} \mathscr{Q}_d}{P|Q \xrightarrow{\overleftrightarrow{s}} \frac{\mathscr{P}|Q \cdot \oplus \mathscr{P}_d}{\oplus \mathscr{P}_d + \oplus \mathscr{Q}_d} + \frac{P|\mathscr{Q} \cdot \oplus \mathscr{P}_d}{\oplus \mathscr{P}_d + \oplus \mathscr{Q}_d} + \frac{(\nu r)(\mathscr{P}_d|\mathscr{Q}_i)}{\oplus \mathscr{P}_d + \oplus \mathscr{Q}_d} + \frac{(\nu r)(\mathscr{P}_i|\mathscr{Q}_d)}{\oplus \mathscr{P}_d + \oplus \mathscr{Q}_d}}$$

Alternative synchronization functions...

Cooperative synchronization (à la TIPP)

In TIPP the synchronization rate of two activities with rate r_1 and r_2 is defined as the product $r_1 \cdot r_2$:

$$\frac{P \xrightarrow{\overleftrightarrow{s}} \mathscr{P} \quad P \xrightarrow{s(r)} \mathscr{P}_d \quad P \xrightarrow{\overline{s}(r)} \mathscr{P}_i \quad Q \xrightarrow{\overleftrightarrow{s}} \mathscr{Q} \quad Q \xrightarrow{s(r)} \mathscr{Q}_d \quad Q \xrightarrow{\overline{s}(r)} \mathscr{Q}_i}{P|Q \xrightarrow{\overleftrightarrow{s}} \mathscr{P}|Q + (\nu r)(\mathscr{P}_d|\mathscr{Q}_i) + (\nu r)(\mathscr{P}_i|\mathscr{Q}_d) + P|\mathscr{Q}}$$

Alternative synchronization functions...

Patient synchronization (à la PEPA)

In PEPA the synchronization rate of two activities is based the notions of apparent and *minimal rates*:

$$\frac{P \xrightarrow{\overleftrightarrow{s}} \mathscr{P} \quad P \xrightarrow{s(r)} \mathscr{P}_d \quad P \xrightarrow{\bar{s}(r)} \mathscr{P}_i \quad Q \xrightarrow{\overleftrightarrow{s}} \mathscr{Q} \quad Q \xrightarrow{s(r)} \mathscr{Q}_d \quad Q \xrightarrow{\bar{s}(r)} \mathscr{Q}_i}{P|Q \xrightarrow{\overleftrightarrow{s}} \left(\frac{\mathscr{P}|Q}{\mathcal{W}[\mathscr{P}_d, \mathscr{P}_i]} + \frac{P|\mathscr{Q}}{\mathcal{W}[\mathscr{Q}_d, \mathscr{Q}_i]} + \mathscr{P}_d|\mathscr{Q}_i + \mathscr{P}_i|\mathscr{Q}_d \right) \cdot \mathcal{W}[\mathscr{P}_d + \mathscr{Q}_d, \mathscr{P}_i + \mathscr{Q}_i]}$$

where $\mathcal{W}[\mathscr{P}, \mathscr{Q}] \stackrel{def}{=} \frac{min(\oplus\mathscr{P}, \oplus\mathscr{Q})}{\oplus\mathscr{P} \cdot \oplus\mathscr{Q}}$

From MarCaSPiS to CTMC

For process $P \in \mathcal{C}$, the CTMC of P, is defined as $CTMC[P] \stackrel{def}{=} (S, \mathcal{R})$ where

- $S \stackrel{def}{=} \{Q | \exists Q' \in Der(\{P\}) \wedge Q = rep[Q']\}$
- For all $P, Q \in S$, $P \xrightarrow{\tau} \mathscr{Q}$, $\forall s.P \xrightarrow{\overleftrightarrow{s}} \mathscr{Q}_s$:

$$\mathcal{R}[P, Q] \stackrel{def}{=} \mathscr{Q}_{/\equiv}(Q) + \sum_s \mathscr{Q}_{s/\equiv}(Q)$$

Determinacy and Finiteness of reductions

- If $P \equiv Q$, for any α, $\mathscr{P}$, $\mathscr{Q}$ such that $P \xrightarrow{\alpha} \mathscr{P}$ and $Q \xrightarrow{\alpha} \mathscr{Q}$: $\mathscr{P}/\equiv = \mathscr{Q}/\equiv$.
- For each P, α and $\mathscr{P}$, if $P \xrightarrow{\alpha} \mathscr{P}$ then $\{Q | \mathscr{P} \neq 0\}$ is finite.

A stochastic equivalence for MarCaSPiS

Rate aware bisimulation

An equivalence relation $\mathcal{R}$ is a *rate aware* bisimulation if and only if, for each P and Q such that $P\mathcal{R}Q$ then for each α, $\mathscr{P}$, $\mathscr{Q}$:

$$P \xrightarrow{\alpha} \mathscr{P} \wedge Q \xrightarrow{\alpha} \mathscr{Q} \Longrightarrow \mathscr{P}/\mathcal{R} = \mathscr{Q}/\mathcal{R}$$

Rate aware equivalence

We let $\sim$ be the largest *rate aware* bisimulation.

Theorem

For each P and Q, if $P \sim Q$ then CTMC[P] and CTMC[Q] are strong markovian equivalent.

CaSPiS Syntax (with close)

P, Q	$::=$	$\sum_{i \in I} \pi_i P_i$	Guarded Sum
	$\mid$	close	Close
	$\mid$	$k \cdot P$	Listener
	$\mid$	$s_k.P$	Service Definition
	$\mid$	$\overline{s_k}.Q$	Service Invocation
	$\mid$	$P > Q$	Pipeline
	$\mid$	$r \rhd_k P$	Session
	$\mid$	$\blacktriangleright P$	Terminated Session
	$\mid$	$P \mid Q$	Parallel Composition
	$\mid$	$(\nu n)P$	Name Restriction
	$\mid$	$!P$	Replication

CaSPiS: Closing sessions

The syntax of service definition and service invocation become $\overline{s_k}.Q$ and $s_k.P$, where k is used for identifying the termination handler service to be associated to the other side of the instantiated session.

$$s_{k_1}.P \xrightarrow{s(r)^{k_2}_{k_1}} r \rhd_{k_2} P \qquad \overline{s}_{k_2} Q \xrightarrow{\overline{s}(r)^{k_1}_{k_2}} r \rhd_{k_1} P$$

$$\frac{P \xrightarrow{s(r)^{k_2}_{k_1}} P' \quad Q \xrightarrow{\overline{s}(r)^{k_1}_{k_2}} Q'}{P|Q \xrightarrow{\tau} (\nu r)(P'|Q')}$$

CaSPiS: Closing sessions

We associate to each session a service name k which identifies a *termination handler* service $(r \rhd_k P)$
As soon as P execute a close action, session r enters a closing state $(\blacktriangleright Q)$; the corresponding termination handler is invoked

$$\text{close} \xrightarrow{\text{close}} \mathbf{0} \qquad \frac{P \xrightarrow{\text{close}} P'}{r \rhd_k P \xrightarrow{\tau} \blacktriangleright P' | \dagger(k)}$$

CaSPiS: Closing sessions

A closing session can only trigger further closing of nested subsessions.

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

Right after execution of close a signal $\dagger(k)$ is sent to the termination-handler service k listening at the *opposite* side of the session:

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

CaSPiS: An Example

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

- Process *News* implements a *news collector* exposing service `collect`.
- After invocation of this service, a client receives all the news produced by `CNN`, `BBC` and `ANSA`.
- The established session can be closed:
  - by the client-side,
  - or, (ii) when any of the three nested sessions used for interacting with the news services is closed by peer

Programming graceful closure

Attention is needed to make sure that sessions are properly closed by providing appropriate notice.

- CaSPiS session closing primitives do not guarantee *per se* that forever-dangling, one-sided sessions, never arise
- This undesirable situation can be avoided if one makes sure that specific termination handlers ($k.C[\mathsf{close}]$), are installed in the bodies of client's invocation and service definition
- $C[\cdot]$ may contain extra actions that the termination handler may wish to take upon invocation, e.g., further signaling to other listeners (a sort of compensation, in the language of long-running transactions).

Some definitions...

Definition

A context $C[\cdot]$ is

- *static* if the hole does not occur in the scope of a dynamic operator
- *session-immune* if its hole does not occur in the scope of a session operator.

Definition (service-level balancing)

We say P is *service-balanced* if for each s and k:

1. $s_k.$ and $\overline{s}_k.$ may only occur in P in subterms of the form $s_k.(C[k \cdot C'[\text{close}]])$ or $\overline{s}_k.(C[k \cdot C'[\text{close}]])$, with $C[\cdot]$, $C'[\cdot]$ static and session immune;
2. there are in P at most one occurrence of the listener $k\cdot$ and one occurrence of $\triangleright_k$.

Main Result

Definition (r-balancing)

Let P be process. We say P is

- *r-balanced* if either $r \notin \mathrm{fn}(P)$ or, up to structural congruence, in P there occur two r-sides that form a balanced pair.

Definition (balanced processes)

Assume $P \equiv (\nu \tilde{r})Q$, where Q has no top bound sessions and $\tilde{r} \subseteq \mathrm{fn}(Q)$. We say P is *balanced* if:

(a) P is service-balanced, and

(b) for each $r \in \mathrm{fn}(Q)$, Q is r-balanced.

Theorem (graceful closure property)

Let P be balanced. Whenever $P \xrightarrow{\tau} {}^{*}P'$ there exists a Q balanced such that $P' \xrightarrow{\tau} {}^{*}Q$.

Thank you for your attention!