

Service Composition Algebra

Simon Foster

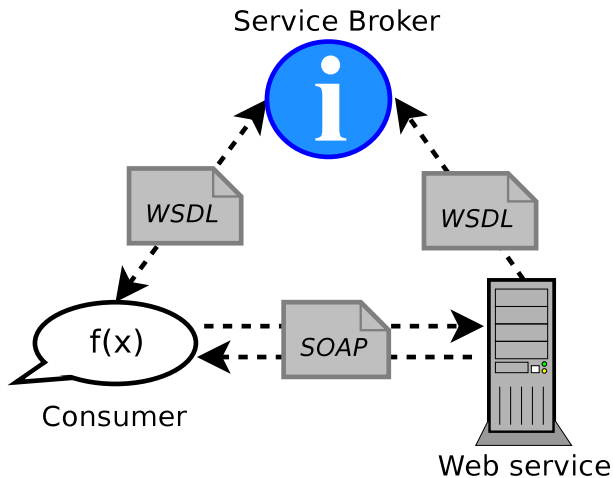
`<S.Foster@dcs.shef.ac.uk>`

Verification and Testing Group
Department of Computer Science
University of Sheffield

October 23, 2007

- ▶ A **software component** with a Web-based API.
- ▶ A provider of services.
- ▶ **XML** as a means of information encapsulation.
- ▶ Typically consist of a set of methods attached to one or more URLs.
- ▶ Self describing – types, behaviour etc.
- ▶ Reactive components for building **distributed applications**.
- ▶ **Dynamic** – new services appearing, old ones disappearing.

Web service architecture



Composing Services

- ▶ Web services are domain specific.
- ▶ A high level goal may require several services.
- ▶ e.g. “I want to book a holiday with these requirements...”
- ▶ Data and protocols may be **heterogeneous**.
- ▶ Need some form of **mediation** between the Web services.
- ▶ Akin to **Object Oriented Programming**.
- ▶ **Semantic Web services** promise to automate composition.

- ▶ Two main perspectives:

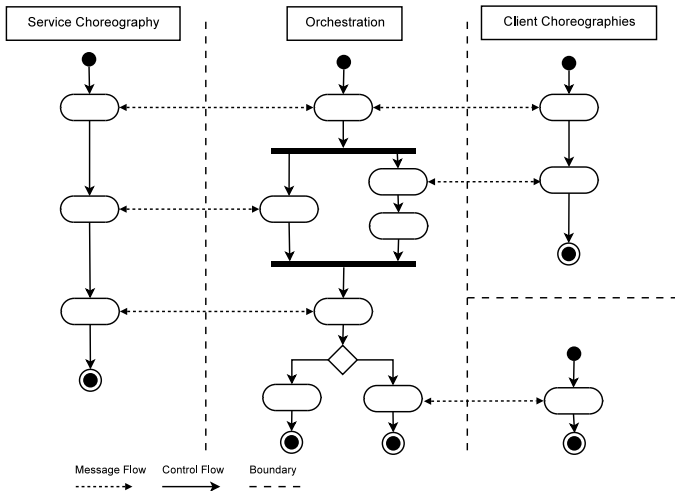
Orchestration – bottom-up, imperative view.

- ▶ How the components are composed to achieve an overall goal.
- ▶ The Web service **implementation**.

Choreography – top-down, declarative view.

- ▶ The protocol for communicating with the Web service.
- ▶ The **behavioural type** of the Web service.

Orchestration and Choreography



Orchestration languages

- ▶ Provide connectives for composing Web services.
- ▶ Need to account for control-flow and data manipulation.
- ▶ Commonly use [workflow patterns](#) for composition.
- ▶ Can be graph based or block-structured.

- ▶ A well recognised orchestration language.
- ▶ Recently **OASIS** standardised at version 2.0
- ▶ Provides a large selection of programming constructs.
- ▶ Supports **long running business transactions**.
- ▶ Also has a large library of tools available.

File Edit Navigate Search Project Run Process Window Help

80% Attend a BPEL Webinar

TestBPEL.bpel

Selection
Marquee
Link
Activity
Operation Wizard
Receive
Reply
Invoke
Assign
Throw
Rethrow
Exit
Wait
Empty
Container
Sequence
If
While
Repeat Until
Pick
Flow
Scope
For Each
Other
Drawing
Custom

Process Variables

BuyBookRequest
BuyBookResponse
Scope
V1

BuyBook Data X
ns2:BuyB...
BuyBo... 20

BuyBook Data X
ns2:BuyB...
BuyBo... run2comp2nowt

V1 Data X
V1 nowt

Process Activities | Fault Handlers | Event Handlers | Source

Debug | Outline

Console | Problems | Properties | Tasks

BuyBookProcess [Simulate Process] ActiveBP

```

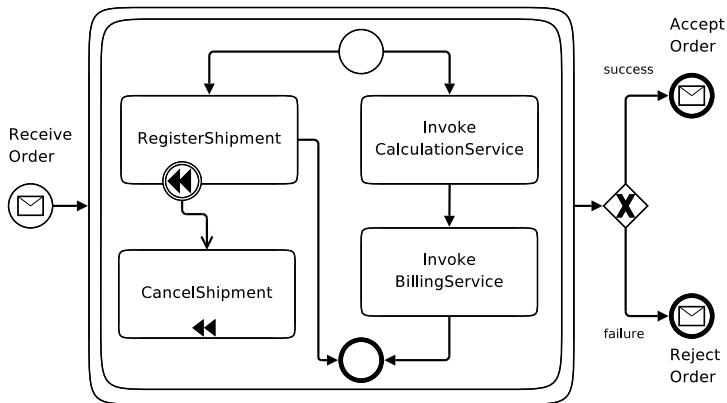
Sequence : Completed normally [/process/sequence/scope/faultHandlers/ca
Catch All : Completed normally [/process/sequence/scope/faultHandlers/c
Scope : Completed normally [/process/sequence/scope]

```

Compensable transactions

- ▶ Important feature of BPEL.
- ▶ **All or nothing** parts of the orchestration.
- ▶ Each activity can be associated with a **compensation**.
 - ▶ Installed upon successful completion of associated activity.
 - ▶ Run in reverse order upon transaction abortion.
- ▶ BPEL also **snapshots** variables.

Example Transaction



- ▶ Primarily **graph** based (cf. Petri nets)
- ▶ A BPEL process becomes a guarded graph (a **flow**)
- ▶ Question marks over **compositionality**
- ▶ Certain features of BPEL may hinder efforts
- ▶ Need a language and semantics with compositionality as a core concept
- ▶ Want to avoid **mobility** (additional complexities) – primarily finite state

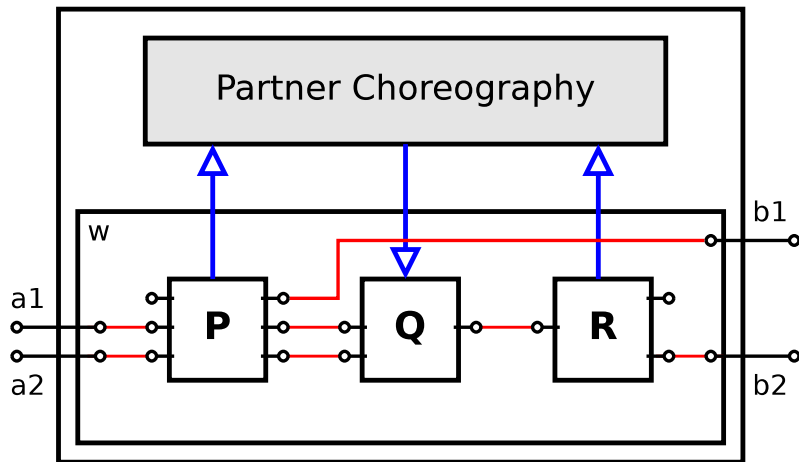
- ▶ To study the algebraic properties of workflow.
- ▶ The algebra will provide the important features of BPEL.
- ▶ An underlying **process algebra**-based semantics to allow behavioural description.
- ▶ Avoiding **mobility** found in existing process algebra solutions.
- ▶ Components decoupled via the use of **dataflow**.
- ▶ On-the-fly verification via **algebraic theory** and **simulation**.

Diagrammatic Level	(UML2AD, BPMN etc.)
Algebraic Level	(Cashew-A)
Process Level	(CaSE+)
Machine Level	(LTS, ASM etc.)

Diagrammatic Level	(UML2AD, BPMN etc.)
Algebraic Level	(Cashew-A)
Process Level	(CaSE+)
Machine Level	(LTS, ASM etc.)

- ▶ An algebra for describing service compositions.
- ▶ Orchestration given in terms of its **control flow** and **data flow**.
- ▶ Interaction defined between workflow and a partner.
- ▶ Attempts to model all the **finite-state** control flow patterns.
- ▶ Includes support for compensation.
- ▶ Loosely based on **Kleene algebra** (Kozen 1990).
- ▶ Uses an electrical component metaphor.
- ▶ **Preconditions** central to determining execution order.

Example



$$\mathcal{W} ::= w[\{\mathcal{A}\}(Cf \times Df)\{\mathcal{B}\}]$$
$$Cf ::= Nary \mid \mathcal{C}$$
$$Df ::= \mathcal{D}$$

- ▶ \mathcal{W} defines a workflow named w .
- ▶ Cf and Df define the control and data flow.
- ▶ \mathcal{A} and \mathcal{B} define the pre and post-conditions of the workflow.
- ▶ These usually define inputs and outputs of the workflow.
- ▶ Input requirements must be satisfied before it can execute.

Control flow algebra

$$\begin{aligned} \mathcal{C} &::= \mathcal{C};\mathcal{C} \mid \mathcal{C} \parallel \mathcal{C} \mid \mathcal{C} \parallel\!\!\parallel \mathcal{C} \mid \mathcal{C} \oplus \mathcal{C} \mid \mathcal{C}^*\mathcal{C} \mid \epsilon \mid \delta \mid \uparrow\mathcal{C} \mid \mathcal{P} \\ \mathcal{P} &::= p[Pf] \end{aligned}$$

- ▶ Similar to [Kleene algebra](#), but with different axioms.
- ▶ Composes [performances](#), named units of behaviour with inputs and outputs.

Algebraic Semantics - Sequence and Choice

$$E, F, G \in \mathcal{C}$$

$$E \ ; \ (F \ ; \ G) = (E \ ; \ F) \ ; \ G$$

$$E \ ; \ \epsilon = E$$

$$\delta \ ; \ E = \delta$$

$$(E \oplus F) \ ; \ G = (E \ ; \ G) \oplus (F \ ; \ G)$$

$$E \oplus (F \oplus G) = (E \oplus F) \oplus G$$

$$E \oplus F = F \oplus E$$

$$E \oplus E = E$$

$$E \oplus \delta = E$$

Algebraic Semantics - Parallel and Yield

$$E \parallel (F \parallel G) = (E \parallel F) \parallel G$$

$$E \parallel F = F \parallel G$$

$$E \parallel \epsilon = E$$

$$E \parallel \delta = \delta$$

$$E \parallel (F \oplus G) = (E \parallel F) \oplus (E \parallel G)$$

$$(E \textcircled{;} \delta) \parallel F = (E \parallel F) \textcircled{;} \delta$$

$$E \parallel\!\!\parallel F = ((\epsilon \textcircled{;} E) \parallel F) \oplus (E \parallel (\epsilon \textcircled{;} F))$$

$$\uparrow E \oplus \uparrow F = \uparrow(E \oplus F)$$

$$\uparrow E \parallel \uparrow F = \uparrow(E \parallel F)$$

$$\uparrow \delta = \delta$$

$$F \oplus \uparrow(E \textcircled{;} E^*F) = E^*F$$

$Nary ::= \text{Inter } PList \mid \text{Race } TList$

- ▶ Some of the important workflow patterns rely on being n-ary.
- ▶ Interleaving runs the collection of processes one at a time without a specific order.
- ▶ Race allows **speculative parallelism**.
 - ▶ Run all the workflows in parallel until one completes, then cancel the others.
 - ▶ Provides a way of achieving a result by multiple means.
 - ▶ Depends on compensation.

$$\mathcal{A} ::= \mathcal{A} \sqcup \mathcal{A} \mid \mathcal{A} \sqcap \mathcal{A} \mid \mathbf{0} \mid \mathbf{1} \mid a$$

$$\mathcal{B} ::= \mathcal{B} \sqcup \mathcal{B} \mid \mathcal{B} \sqcap \mathcal{B} \mid \mathbf{0} \mid \mathbf{1} \mid b$$

$$\mathcal{D} ::= a \triangleright p.a \cdot \mathcal{D} \mid p.b \triangleright p.a \cdot \mathcal{D} \mid p.b \triangleright b \cdot \mathcal{D} \mid \mathbf{1}$$

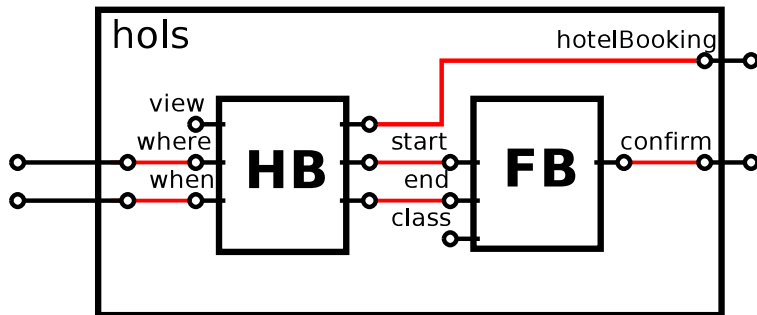
- ▶ \mathcal{A} and \mathcal{B} are **Boolean algebras** defining workflow pre- and post-conditions.
- ▶ They are used to specify inputs and outputs.
- ▶ $X \sqcup Y$ means either X or Y should be satisfied.
- ▶ $X \sqcap Y$ means both X and Y should be satisfied.
- ▶ \mathcal{D} describes wires between inputs, performances and outputs.
- ▶ \mathcal{D} 's \cdot is **commutative** and **idempotent**.

$\text{flatNo?} \sqcap \text{houseNo} \sqcap (\text{postcode} \sqcup (\text{street} \sqcap \text{city}))$
(where $a? = a \sqcup \mathbf{1}$)

- ▶ This input expression would specify all the possibilities required for an address.
- ▶ Can be satisfied by

$\{\{\text{houseNo}, \text{postcode}\}, \{\text{flatNo}, \text{houseNo}, \text{postcode}\},$
 $\{\text{houseNo}, \text{street}, \text{city}\} \dots \}$

Holiday Booker



- ▶ Data flow can be verified by **Weakest Liberal Preconditions** (Bonsangue, Kok 1994).
- ▶ From every postcondition can we prove there exists a corresponding precondition?
- ▶ Control flow + Data flow should define a **total relation** from post to preconditions.

i.e. if $w[\{A\}(C \times D)\{B\}]$ then $C \times D \vdash B \rightarrow A$

- ▶ First we need to define a predicate for the workflow state

$$\text{DfS} ::= \text{DfS} + \text{DfS} \mid \text{DfS} \cdot \text{DfS} \mid \mathbf{0} \mid \mathbf{1} \mid a^p \mid b^p$$

- ▶ \mathcal{D} gives rise to a function $\text{dft} : \mathcal{D} \rightarrow (\text{DfS} \rightarrow \text{DfS})$ mapping wires to a pre – post-condition transformer
- ▶ e.g. $D = p.b \triangleright q.a \cdot r.b \triangleright q.a \cdot \mathbf{1} \implies (a^q \mapsto b^p + b^r) \in \text{dft}(D)$
- ▶ Unmapped preconditions become $\mathbf{0}$
- ▶ Each performance also defines a post – pre-condition transformer.

Weakest Liberal Precondition

- ▶ Under the context of a particular \mathcal{D} we can define wlp :

$$\begin{aligned}wlp & : \mathcal{C} \times \text{DfS} \rightarrow \text{DfS} \\wlp(\epsilon, R) & = R \\wlp(\delta, R) & = \mathbf{0} \\wlp(P \text{ ; } Q, R) & = wlp(P, dft(\mathcal{D}, wlp(Q, R))) \\wlp(P \oplus Q, R) & = wlp(P, R) + wlp(Q, R) \\wlp(P \parallel Q, R) & = wlp(P, R) \cdot wlp(Q, R) \\wlp(P^* Q, R) & = wlp(P, dft(\mathcal{D}, wlp(Q, R))) \\wlp(\uparrow P, R) & = wlp(P, R)\end{aligned}$$

- ▶ Verification fails if wlp evaluates to $\mathbf{0}$.

- ▶ We need 2 relations for components:
 1. A given component's context must provide more inputs and require less outputs than is provided (but must also work for all options).
 2. A given component must
 - ▶ Provide implementations for each of the possible sub-expressions of the input;
 - ▶ Provide at least one of the output possibilities.
- ▶ May be possible to define in terms of **wlp**

$Pf ::= \mathcal{W} \mid \mathcal{T} \mid \textit{Goal} \mid \textit{Eval} \mid \textit{Send} \mid \textit{Receive} \mid \textit{Coordinate}$

$\textit{Send} ::= \mathbf{Send} \ m \ \tilde{a}$

$\textit{Receive} ::= \mathbf{Receiver} \ m \ \tilde{b}$

$\textit{Coordinate} ::= \mathbf{Coord} \ \textit{Partner} \ \mathcal{W}$

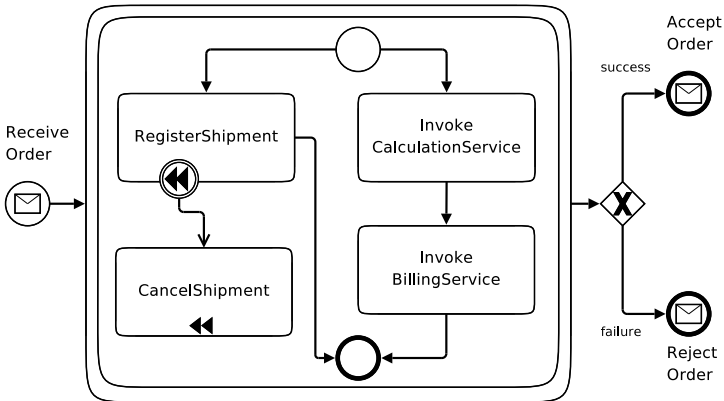
- ▶ *Eval* evaluates an expression, acting as δ if false, otherwise producing outputs.
- ▶ *Send* and *Receive* messages to the associated partner.
- ▶ *Receive* is predicated on the message receipt.
- ▶ *Coordinate* composes a partner with a workflow. Closes message interaction context.

Compensable Transactions

$$\begin{aligned}\hat{C} &::= \hat{C};\hat{C} \mid \hat{C}\parallel\hat{C} \mid \hat{C}\oplus\hat{C} \mid \hat{C}\div\hat{C} \mid \epsilon \mid \delta \mid \downarrow \mid \uparrow\hat{C} \mid \uparrow\hat{C} \mid \mathcal{P} \\ \mathcal{T} &::= \mathbf{T} \hat{W} \\ \hat{W} &::= w[\{\mathcal{A}\}(\hat{C}f \times Df)\{\mathcal{B}\}]\end{aligned}$$

- ▶ A workflow can be enclosed within a transaction.
- ▶ The compensations will be executed in reverse order to their installation upon exception (\downarrow).
- ▶ By virtue of the dataflow model, each compensation need not be concerned with variable “snapshots”.

Example Transaction



$$\text{RecOrder} \wp \text{trans}[\mathbf{T} \text{ wfOrder}[\mathbf{1}\{\text{RegShipment} \div \text{CancelShipment} \\ \parallel \text{CalcBill} \wp \text{SendBill} \times \mathbf{1}\}\mathbf{1}]] \\ \wp (\text{EvalSuccess} \wp \text{SendAccept} \oplus \text{EvalFailure} \wp \text{SendReject})$$

Diagrammatic Level	(UML2AD, BPMN etc.)
Algebraic Level	(Cashew-A)
Process Level	(CaSE+)
Machine Level	(LTS, ASM etc.)

Timed Process Algebra

- ▶ Process descriptions based on some form of temporal device.
- ▶ Prioritisation helps resolve non-determinism.
- ▶ Split into **Real Time** and **Abstract Time**.
- ▶ Latter treats time **qualitatively** – an abstract event **modelling** time.
- ▶ We give a semantics to Cashew-A using such a calculus.

Calculus of Synchronisation and Encapsulation

- ▶ **CaSE** is a conservative extension of **CCS**.
- ▶ Time as a discrete multi-party event.
- ▶ Individual **clocks** scoped to different regions of a process.
- ▶ Clocks are **hidden**, cf. **CSP**.
- ▶ Clock **ticks** (σ or ρ) pre-empted by internal actions.
- ▶ All internal activity must be exhausted before a tick is observed – **maximal progress**.
- ▶ **CaSE+** additionally draws ideas from **TCCS** and **CBS**.
- ▶ Equipped with a semantic congruence (also axiomatised).

Why abstract time to model workflow?

1. Facilitates scalable data flow – **isochronic broadcast**.
2. Permits elegant modelling of **yield**.
3. Provides an elegant method of **exception handling**.
4. Block structure fits neatly into **synchronous hierarchies**.

$$\begin{aligned} \mathcal{E} ::= & \mathbf{0} \mid \mathbf{1} \mid \alpha.\mathcal{E} \mid \underline{\sigma}.\mathcal{E} \mid \neg\sigma.\mathcal{E} \mid \mathcal{E} + \mathcal{E} \mid \mathcal{E} ++ \mathcal{E} \mid \mathcal{E}|\mathcal{E} \mid \mathcal{E} \setminus a \\ & \mid \mathcal{E}[a \mapsto a] \mid \mathcal{E}[\sigma \mapsto a] \mid \mathcal{E}/\sigma \mid \mu X.\mathcal{E} \mid X \end{aligned}$$

- ▶ Standard CCS operators

$$\begin{aligned} \mathcal{E} ::= & \mathbf{0} \mid \mathbf{1} \mid \alpha.\mathcal{E} \mid \underline{\sigma}.\mathcal{E} \mid \neg\sigma.\mathcal{E} \mid \mathcal{E} + \mathcal{E} \mid \mathcal{E} ++ \mathcal{E} \mid \mathcal{E}|\mathcal{E} \mid \mathcal{E}\backslash a \\ & \mid \mathcal{E}[a \mapsto a] \mid \mathcal{E}[\sigma \mapsto a] \mid \mathcal{E}/\sigma \mid \mu X.\mathcal{E} \mid X \end{aligned}$$

- ▶ Standard CCS operators
- ▶ Idle, Clock prefix, Negative clock prefix (any clock but)

$$\begin{aligned} \mathcal{E} ::= & \mathbf{0} \mid \mathbf{1} \mid \alpha.\mathcal{E} \mid \underline{\sigma}.\mathcal{E} \mid \neg\sigma.\mathcal{E} \mid \mathcal{E} + \mathcal{E} \mid \mathcal{E} ++ \mathcal{E} \mid \mathcal{E}|\mathcal{E} \mid \mathcal{E}\backslash a \\ & \mid \mathcal{E}[a \mapsto a] \mid \mathcal{E}[\sigma \mapsto a] \mid \mathcal{E}/\sigma \mid \mu X.\mathcal{E} \mid X \end{aligned}$$

- ▶ Standard CCS operators
- ▶ Idle, Clock prefix, Negative clock prefix (any clock but)
- ▶ Weak clock choice, Rename clocks to actions, Clock hiding

- ▶ Equivalence theory based on **weak bisimulation**.
- ▶ Provides a basis for checking choreography **conformance**.
- ▶ Actual behavioural relation called **Alternating Simulation**.

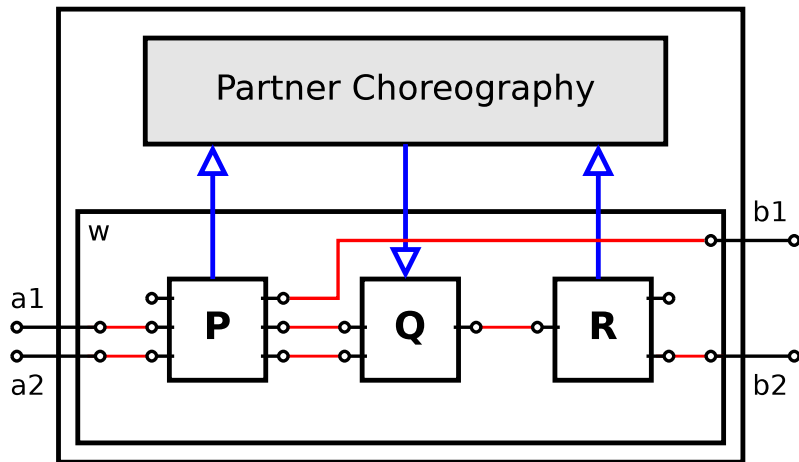
Definition

Temporal Weak Bisimulation (t.w.b.) A symmetric relation \mathcal{R} is a t.w.b. provided $\forall (E, F) \in \mathcal{R}$:

- ▶ If $E \xrightarrow{\gamma} E'$ then $\exists F'. F \xrightarrow{\hat{\gamma}} F'$ and $(E', F') \in \mathcal{R}$

We write $E \approx F$ if $(E, F) \in \mathcal{R}$ for any t.w.b.

Example



Definition

Temporal Observation Congruence (t.o.c.)

A symmetric relation \mathcal{R} is a t.o.c. provided $\forall (E, F) \in \mathcal{R}$:

1. If $E \xrightarrow{\alpha} E'$ then $\exists F'. F \xrightarrow{\alpha} F'$ and $E' \approx F'$
2. If $E \xrightarrow{\sigma} E'$ then $\exists F'. F \xrightarrow{\sigma} F'$ and $(E', F') \in \mathcal{R}$

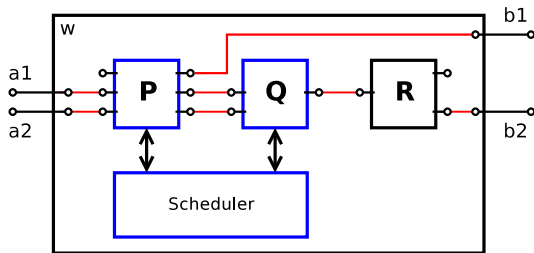
We write $E \cong F$ if $(E, F) \in \mathcal{R}$ for any t.o.c.

Theorem

Compositionality

Temporal Observation Congruence is compositional through all operators of CaSE+.

Workflow Scheduling



- ▶ A workflow will be represented as a composition of processes.
- ▶ Each construct will be allocated a scheduler process.
- ▶ Schedulers are then composed with the workflow semantics.
- ▶ e.g. $\llbracket W \parallel V \rrbracket = \llbracket W \rrbracket \mid \llbracket \text{Scheduler} \rrbracket \mid \llbracket V \rrbracket$
- ▶ Schedulers will order execution.

- ▶ Each component is scheduled via common protocol.

\xrightarrow{e}

- ▶ e – permission to execute

Scheduling Protocol

- ▶ Each component is scheduled via common protocol.

$$\xrightarrow{\bar{r}} \quad \xrightarrow{e}$$

- ▶ e – permission to execute
- ▶ r – indication of readiness to execute

Scheduling Protocol

- ▶ Each component is scheduled via common protocol.

$$\xrightarrow{g} \xrightarrow{\bar{r}} \xrightarrow{e}$$

- ▶ e – permission to execute
- ▶ r – indication of readiness to execute
- ▶ g – permission to fulfil preconditions

Scheduling Protocol

- ▶ Each component is scheduled via common protocol.

$$\xrightarrow{g} \xrightarrow{\bar{r}} \xrightarrow{e} \xrightarrow{\bar{\checkmark}}$$

- ▶ e – permission to execute
- ▶ r – indication of readiness to execute
- ▶ g – permission to fulfil preconditions
- ▶ \checkmark – execution complete

- ▶ Each expression in Cashew-A is mapped to a CaSE+ process.
- ▶ A semantic function gives a CaSE+ denotation to each element of Cashew-A.

$$\llbracket - \rrbracket : \text{Construct} \times \text{Identifier} \times \text{Expression} \times \tilde{a} \times \tilde{b} \rightarrow \mathcal{E}$$

Sequential Composition Semantics

Definition

$$\mathbf{wf}_w \llbracket P \circledast Q \rrbracket_{\tilde{b}^P \cup \tilde{b}^Q}^{\tilde{a}^P \cup \tilde{a}^Q} =$$

$$\left(\mathbf{wf}_w \llbracket P \rrbracket_{\tilde{b}^P}^{\tilde{a}^P} [\mathfrak{F}] \mid \mathbf{wf}_w \llbracket Q \rrbracket_{\tilde{b}^Q}^{\tilde{a}^Q} [\mathfrak{G}] \mid \mu X. g. \bar{g}^l. (r^l. \bar{r}. e. \bar{e}^l. \checkmark^l. \bar{g}^r. r^r. \bar{e}^r. \checkmark^r. \bar{\checkmark}. X \triangleright_e s. \bar{s}^l. X) \right) \setminus \mathfrak{R}$$

- ▶ Channels of left and right workflows (g, r, e, \checkmark) renamed via \mathfrak{F} and \mathfrak{G}
- ▶ These channels are then used to orchestrate the processes.
- ▶ Then they are restricted via \mathfrak{R} .
- ▶ Inputs and outputs of both also collected.

- ▶ Each workflow is split up into a number of distinct **phases**.
- ▶ A phase is the activity which occurs between two clock ticks.
- ▶ Normal workflow activity is divided into 5 phases, delimited by $\sigma_1^w \cdots \sigma_5^w$.
- ▶ Processes in the workflow lock-step through each phase.
- ▶ e.g. Workflow becomes ready to execute (σ_2^w) when
 - ▶ All sub-components are ready;
 - ▶ Dataflow agents allow clock to tick.
- ▶ Clocks are scoped to their respective workflows.

Diagrammatic Level	(UML2AD, BPMN etc.)
Algebraic Level	(Cashew-A)
Process Level	(CaSE+)
Machine Level	(LTS, ASM etc.)

data Process $a p v m$

$$\begin{aligned} &= \underline{\text{NIL}} \\ &| \underline{\text{OBS}} (\text{Action } a v m) (\text{Process } a p v m) \\ &| \underline{\text{TAU}} (m \text{ Int}) [\text{Process } a p v m] \\ &| \underline{\text{SUM}} (\text{Process } a p v m) (\text{Process } a p v m) \\ &| \underline{\text{PAR}} (\text{Process } a p v m) (\text{Process } a p v m) \end{aligned}$$

data Action $a v m$

$$= \underline{\text{Input}} a h (v \rightarrow m ()) \mid \underline{\text{Output}} a h (m v) \mid \underline{\text{Silent}}$$

- ▶ Basic CCS processes, cut down from CaSE+ implementation.
- ▶ Inputs/Outputs have monadic actions to give them real-world behaviour.

$step :: \text{Monad } m \Rightarrow \text{Process } a \ p \ v \ m \rightarrow [\text{Transition } a \ p \ v \ m]$

$runProcess :: \text{MonadIO } m \Rightarrow \text{Process } a \ p \ v \ m \rightarrow m \ ()$

- ▶ We can then execute the process using the above function.

Advantages of my approach

1. Components are **fully separated** from their environment, maximising reuse possibilities;
2. Dataflow seems more natural to Web services than mutable variables;
3. Simplified interaction model avoid the complexities of **π -calculus**;
4. We can employ existing simulation techniques to verify workflows;
5. We can employ various algebraic reasoning techniques;
6. We get a client–server choreography for free;
7. A standard architecture opens the door for flexible addition of control flow patterns.

- ▶ Compensation can itself be seen as following design patterns. e.g. Decentralised vs. Centralised compensation.
- ▶ Want to investigate algebraic properties of compensation.
- ▶ Need to formalise my method of model checking.
- ▶ An implementation of a Cashew-A server is planned in Haskell. CaSE+ is already implemented.

Conclusions

- ▶ I have presented an algebra for Web service orchestration.
- ▶ The language provides many workflow patterns, compensation and links choreography and orchestration.
- ▶ Semantics are given using a timed process algebra.

- ▶ Diane Jordan (chair) *et al.* Web Services Business Process Execution Language Version 2.0. **OASIS Standard**. <http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html>
- ▶ W.M.P van der Aalst *et al.* Workflow Patterns. **Distributed and Parallel Databases**, 14(3), pp 5–51, July 2003
- ▶ Barry Norton *et al.* A Compositional Semantic Theory for Synchronous Component-Based Design. **CONCUR 2003**. LNCS 2761.

$$\mathcal{A} = \Lambda \cup \bar{\Lambda} \cup \{\tau\} \quad \mathcal{T} = \{\sigma, \rho \dots\}$$

$$\alpha, \beta \in \mathcal{A} \quad \gamma, \delta \in \mathcal{A} \cup \mathcal{T} \quad E, F, G \dots \in \mathcal{E}$$

$$\begin{aligned} \mathcal{E} ::= & \mathbf{0} \mid \mathbf{1} \mid \alpha.\mathcal{E} \mid \underline{\sigma}.\mathcal{E} \mid \neg\sigma.\mathcal{E} \mid \mathcal{E} + \mathcal{E} \mid \mathcal{E} ++ \mathcal{E} \mid \mathcal{E}|\mathcal{E} \mid \mathcal{E} \setminus a \\ & \mid \mathcal{E}[a \mapsto a] \mid \mathcal{E}[\sigma \mapsto a] \mid \mathcal{E}/\sigma \mid \mu X.\mathcal{E} \mid X \end{aligned}$$

Structural Operational Semantics

$$\text{Act} \quad \frac{-}{\alpha.E \xrightarrow{\alpha} E}$$

$$\text{tTick} \quad \frac{-}{\underline{\sigma}.E \xrightarrow{\sigma} E}$$

$$\text{Sum1} \quad \frac{E \xrightarrow{\alpha} E'}{E + F \xrightarrow{\alpha} E'}$$

$$\text{tSum1} \quad \frac{E \xrightarrow{\sigma} E'}{E + F \xrightarrow{\sigma} E'} \quad a, b$$

$$\text{Sum2} \quad \frac{F \xrightarrow{\alpha} F'}{E + F \xrightarrow{\alpha} F'}$$

$$\text{tSum2} \quad \frac{F \xrightarrow{\sigma} F'}{E + F \xrightarrow{\sigma} F'} \quad a, c$$

$$\text{Sum3} \quad \frac{E \xrightarrow{\alpha} E'}{E ++ F \xrightarrow{\alpha} E'}$$

$$\text{tSum3} \quad \frac{E \xrightarrow{\sigma} E', F \xrightarrow{\sigma} F'}{E + F \xrightarrow{\sigma} E' + F'}$$

$$\text{Sum4} \quad \frac{F \xrightarrow{\alpha} F'}{E ++ F \xrightarrow{\alpha} F'}$$

$$\text{tSum4} \quad \frac{E \xrightarrow{\sigma} E', F \xrightarrow{\sigma} F'}{E ++ F \xrightarrow{\sigma} E' + F'}$$

$$a) \quad E + F \xrightarrow{\tau} \quad b) \quad F \xrightarrow{\sigma} \quad c) \quad E \xrightarrow{\sigma}$$

Structural Operational Semantics

$$\text{tStall} \quad \frac{-}{\neg\sigma.E \xrightarrow{\rho} E} 1$$

$$\text{tPatient} \quad \frac{-}{a.E \xrightarrow{\sigma} a.E}$$

$$\text{tIdle} \quad \frac{-}{\mathbf{1} \xrightarrow{\sigma} \mathbf{1}}$$

$$\text{tCom} \quad \frac{E \xrightarrow{\sigma} E', F \xrightarrow{\sigma} F'}{E|F \xrightarrow{\sigma} E'|F'} d$$

$$\text{Com1} \quad \frac{E \xrightarrow{\alpha} E'}{E|F \xrightarrow{\alpha} E'|F}$$

$$\text{Com3} \quad \frac{E \xrightarrow{a} E', F \xrightarrow{\bar{a}} F'}{E|F \xrightarrow{\tau} E'|F'}$$

$$\text{Res} \quad \frac{E \xrightarrow{\gamma} E'}{E \setminus a \xrightarrow{\gamma} E' \setminus a} 2$$

$$\text{Rel} \quad \frac{E \xrightarrow{\gamma} E'}{E[f] \xrightarrow{f(\gamma)} E'[f]}$$

1) $\rho \neq \sigma$ 2) $\gamma \notin \{a, \bar{a}\}$ d) $E|F \xrightarrow{\sigma}$

$$\text{Hid} \quad \frac{E \xrightarrow{\alpha} E'}{E/\sigma \xrightarrow{\alpha} E'/\sigma}$$

$$\text{tHid1} \quad \frac{E \xrightarrow{\sigma} E'}{E/\sigma \xrightarrow{\tau} E'/\sigma}$$

$$\text{tHid2} \quad \frac{E \xrightarrow{\rho} E'}{E/\sigma \xrightarrow{\rho} E'/\sigma} \quad 1, c$$

$$1) \quad \rho \neq \sigma \quad c) \quad E \not\xrightarrow{\sigma}$$

$$\begin{aligned}
 & \mathbf{wf}_w \llbracket w[A\{C \times D\}B] \times M_i \times M_o \rrbracket_b^a = \\
 & ((\mathbf{ac}_w \llbracket A \rrbracket_{\tilde{a}} | \mu X. (r. \sigma_2^w. \sigma_3^w. X \triangleright_{\sigma_3^w} \psi^w. X)) \setminus r \\
 & | (\mathbf{of}_w \llbracket B \rrbracket_{\tilde{b}} | \mu X. \sigma_4^w. r. \sigma_5^w. X) \setminus r \\
 & | ((\mathbf{cf}_w \llbracket C \rrbracket_{\tilde{b}^{\text{cf}}} \setminus \{a | a \in \tilde{a}^{\text{cf}} \wedge a \notin \tilde{a}^{\text{df}}\} \cup \{b | b \in \tilde{b}^{\text{cf}} \wedge b \notin \tilde{b}^{\text{df}}\}) [\mathfrak{F}] \\
 & \quad | \mu X. \overline{g}. \overline{g}^i. \sigma_1^w. r^i. \sigma_2^w. \overline{r}. e. \sigma_3^w. \overline{e}^i. \mu Y. (\psi^w. Y + \checkmark^i. \sigma_4^w. \sigma_5^w. \overline{\checkmark}. X) \triangleright_e s. \overline{s}^i. \psi^w. X \\
 & \quad) \setminus \{g^i, r^i, e^i, s^i, \checkmark^i\} \\
 & | \mathbf{df}_w \llbracket D \rrbracket_{\tilde{b}^{\text{df}}} [a \mapsto a^w | a \in \tilde{a}] [b \mapsto b^w | b \in \tilde{b}] \setminus \tilde{a}^{\text{df}} \cup \tilde{b}^{\text{df}} \\
 &) \setminus \{a^w | a \in \tilde{a}\} \cup \{b^w | b \in \tilde{b}\} / \{\sigma_1^w \cdots \sigma_5^w, \psi^w\} \\
 & [\sigma^m \mapsto m | m \in \llbracket M_i \rrbracket] [\sigma^m \mapsto \overline{m} | m \in \llbracket M_o \rrbracket]
 \end{aligned}$$

Definition

$$\begin{aligned}\mathbf{ac}_w[A + A']^{\tilde{a} \cup \tilde{a}'} &= ((\mathbf{ac}_w[A]^{\tilde{a}} | \mathbf{ac}_w[A']^{\tilde{a}'})[r \mapsto r^i] \\ &\quad | \mu X. \sigma_1^w. r^i. \bar{r}. [r^i. \sigma_2^w. X] \sigma_2^w(X)) \setminus r^i \\ \mathbf{ac}_w[A \cdot A']^{\tilde{a} \cup \tilde{a}'} &= ((\mathbf{ac}_w[A]^{\tilde{a}} | \mathbf{ac}_w[A']^{\tilde{a}'})[r \mapsto r^i] \\ &\quad | \mu X. \sigma_1^w. (r^i. r^i. \bar{r}. \sigma_2^w. X \triangleright_{\bar{r}} \sigma_2^w. X)) \setminus r^i \\ \mathbf{ac}_w[\mathbf{0}]^{\emptyset} &= \mathbf{1} \\ \mathbf{ac}_w[\mathbf{1}]^{\emptyset} &= \mu X. \sigma_1^w. \bar{r}. \sigma_2^w. X \\ \mathbf{ac}_w[a]^{\{a\}} &= \mu X. \sigma_1^w. (a. \bar{r}. \mu Y. \overline{a^w}. Y \triangleright \sigma_2^w. X)\end{aligned}$$

- ▶ Dataflow is provided by **isochronic broadcast**.