A Formal Model for Web-Service Composition

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

Department of Computer Science
University of Sheffield
http://www.dcs.shef.ac.uk/~simonf

BCTCS 2006

Outline

Outline

Service Composition

- A "Web-Service" is in reality not a service, as a service is temporal, but rather a *service provider* using the Web as a vehicle. The web-service delivers a service.
- Can be thought of as an application with a web-based API.
- Web-Services provide domain specific functionality - in order to provide useful applications we will often need to employ the services of multiple parties.
- The vision is that services could be automatically composed to fulfil any conceivable requirement the user could have.
- For example "I need to stay in Paris for 3 nights, could you book me travel and accommodation with such and such a requirements."

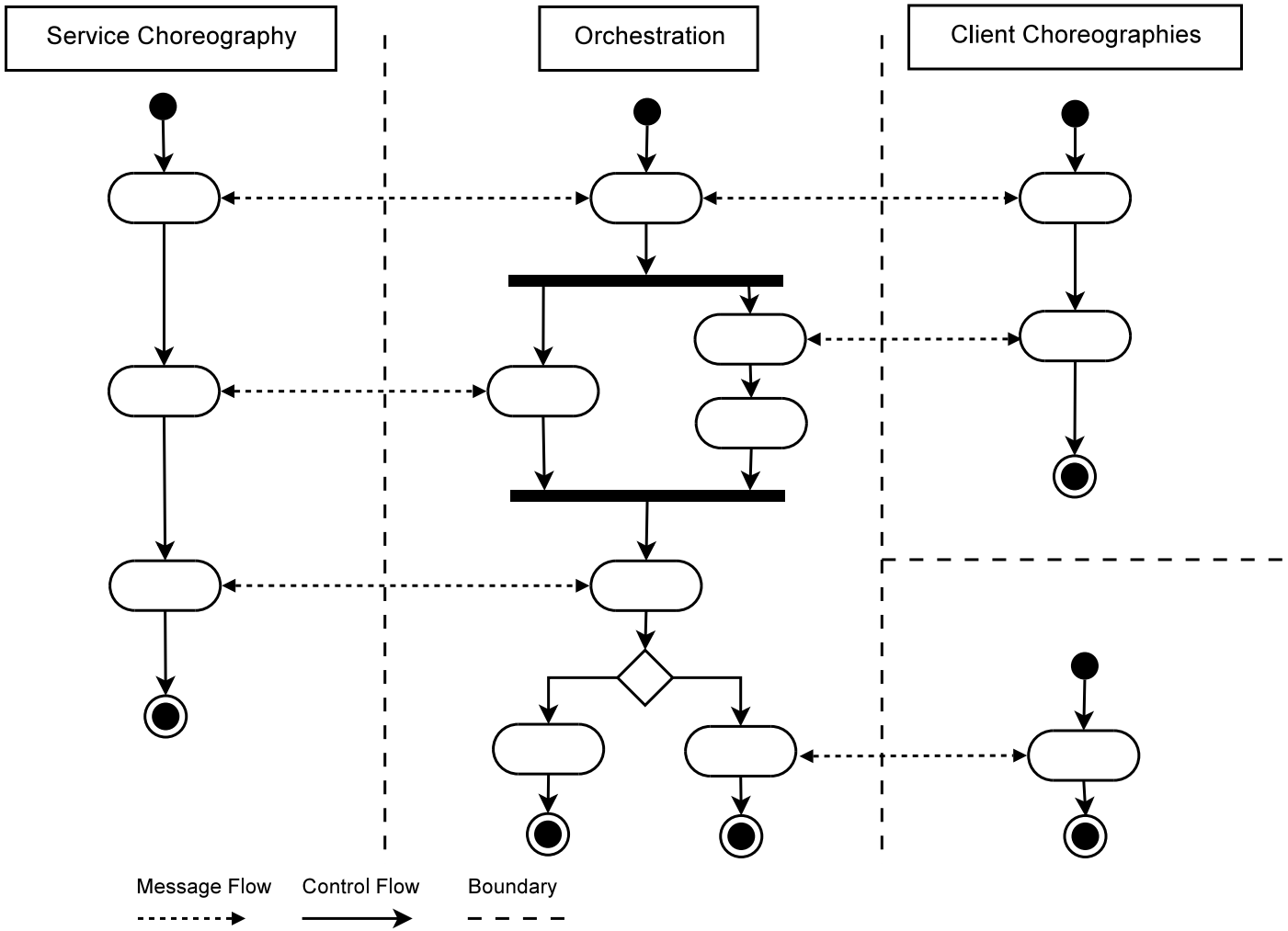
Service Composition (cont.)

- Currently such a query would require browsing several web-sites finding information. It would be better if it could be automated.
- Thus so called *Semantic Web-Services* are emerging which describe themselves with meta-data, allowing other parties to see if they fulfil a particular need they have.
- First we need to establish suitable languages for service description which can make automatic composition possible.
- We need to define methods of service interaction which will allow us to build *Composite Web-Services*.
- The two complementary notions for defining how a composite web-service interacts are *orchestration* and *choreography*.

Components of Services

- **Orchestration** is a bottom-up, imperative view of a service.
- It defines which services need invoking to achieve the composite service's goal and in what order.
- Workflow patterns are frequently, though not always, used.
- By contrast **Choreography** is a top-down, declarative view of a service.
- It defines an agreed method by which parties can communicate and the goal achieved.
- From the point of view of a service, can either be the choreography of another service which we use, or our own choreography.
- May be some sort of state machine model.

A Composite Web-Service



What are the key features?

- WS-BPEL provides important features like *compensable transactions*, but has a questionable theoretical background.
- OWL-S, one of the first Semantic Web-Services languages provides a formal orchestration model, with a sensible subset of core features (workflow oriented), but no choreography.
- The Web Service Modelling Ontology [D. Roman et al., 2005] (WSMO), the most active emerging SWS standard demands both a choreography and an orchestration.
- We need to provide a composition language with both choreography and orchestration which can be given a *compositional* operational semantics.

Outline

The Cashew Project

`http://kmi.open.ac.uk/technologies/cashew`

- Started in 2004 as a Masters project with the aim of giving a formal semantics to web-service composition.
- Continuing as a collaborative effort between Sheffield and KMi.
- Models Web-Services as process algebraic agents.
- Main achievement so far is an orchestration language with a compositional operational semantics.
- Aim is to give a complete composition language with core BPM features.

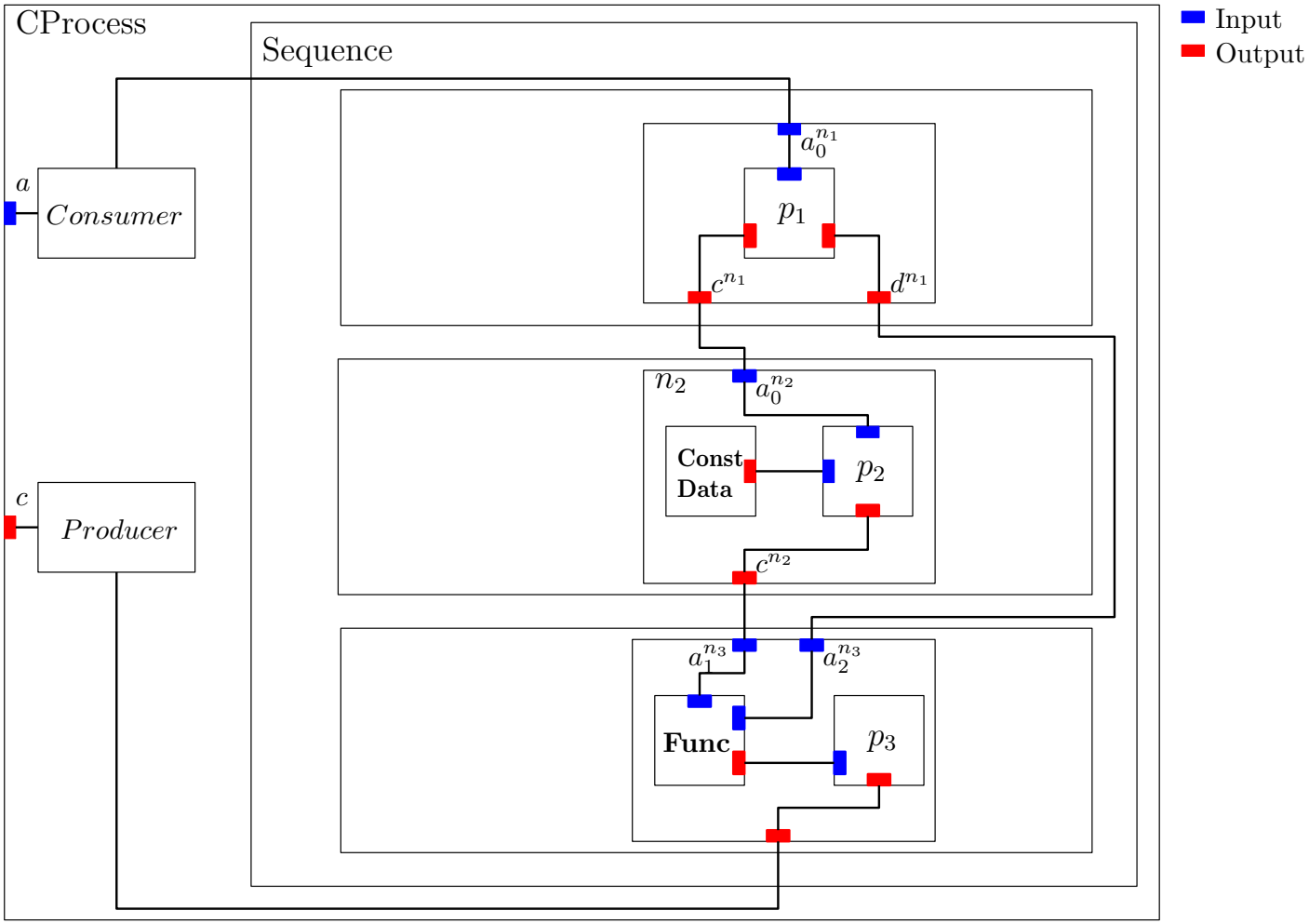
The Cashew-S language

- A language for expressing how services are orchestrated.
- Primarily based on OWL-S, but with a more general model of data-flow where processes explicitly declare where their inputs are going and outputs are coming from.
- Designed with composability and the ability to give it a formal semantics in mind.

Cashew-S - Key Features

- *Processes* which can be either
  - **Atomic** - abstraction of web-service calls.
  - **Composite** - composition of instantiated processes via workflow patterns.
- *Performances*, instantiated processes from which composite processes are made. Allow constant data and functions to be applied to inputs.
- *Consumers* and *Producers*, which connect inputs and outputs from a composite process to the inputs and outputs of the aggregate performances.

Sequential Composition Example



Introducing Cashew-Nuts

- We give this language a semantics using *Cashew-Nuts* [Norton, Foster and Hughes, 2005].
- A timed process calculus based on CCS and descended from Hennessy's *Temporal Process Language* (TPL), and direct descendant of the *Calculus of Synchrony and Encapsulation* [Norton, Lüttgen and Mendler, 2003] (CaSE).
- Uses abstract clocks to facilitate *multi-party synchronisation* - if all composed processes do not prevent a clock from ticking it will do so.

Cashew-Nuts (cont.)

- Originally stood for the *Calculus of Synchronous Hierarchies Extended with Non-deterministic and Untimed Synchronisations*
- Clock ticking can be held up either by explicit prevention or via the presence of silent actions (maximal progress).
- Clocks can either tick black (respecting maximal progress), or tick red (forgoing maximal progress).
- Equivalence theory based on *temporal observation congruence* (CCS's observation congruence with time adaptation).

Cashew-Nuts - Partial Syntax

Definition

$$\mathcal{E} ::= \mathbf{0} \mid \Delta \mid \Delta_\sigma \mid \alpha.\mathcal{E} \mid \lfloor\mathcal{E}\rfloor\sigma(\mathcal{E}) \mid \lceil\mathcal{E}\rceil\sigma(\mathcal{E}) \mid \mathcal{E}+\mathcal{E} \mid \mathcal{E}|\mathcal{E} \mid \mathcal{E}[a \mapsto b] \mid \mathcal{E} \setminus a \mid \mathcal{E}/\sigma \mid \mathcal{E}/\!\!/\sigma \mid \mu X.\mathcal{E} \mid X$$

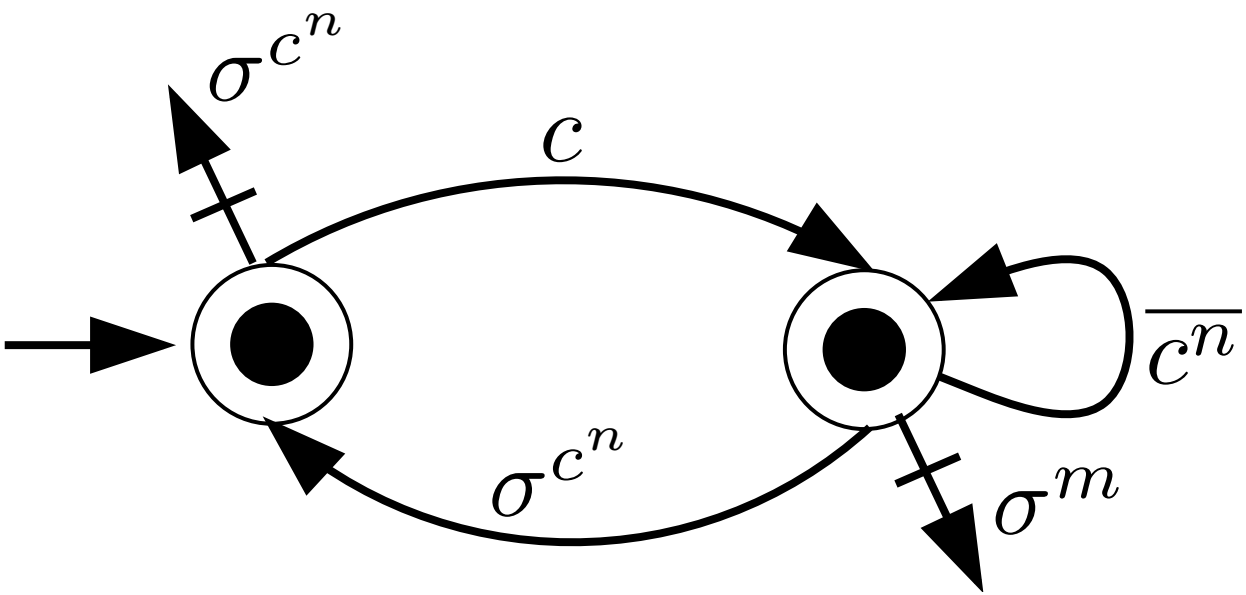
(a is a non-silent action, α is any action, σ is a clock)

- Timeout, behave like LHS if it can act, otherwise tick σ and behave like RHS.
- Δ operator prevents clock(s) from ticking. Summation with an action prefix causes *insistence* (Represented by underlining viz. $\underline{a}.P$).
- Clock Hiding - ticks become silent actions, allows hierarchy. Former version filters red ticks.

Cashew-Nuts for orchestration semantics

- Cashew-Nuts handles control and data flow in Cashew-S compositionally.
- Workflow patterns compositional through temporal observation congruence.
- Mobility is not required, as is generally not relevant for workflow [van der Aalst, 2005].
- Maximal progress overriding is motivated by processes which have "completed" (i.e. the remaining workflow can continue) when all parts have been scheduled, but still have internal activity remaining (e.g. Split).
- May also be required for *exceptions* and *compensable transactions*.

Cashew-Nuts Broadcast



$$\mu X.\underline{c}_{\sigma^{c^n}}.\mu Y.\lfloor c^n.Y \rfloor \sigma^{c^n}(X)$$

- Broadcast allows data to be sent to everyone who requires it, terminating afterward.

Cashew-Nuts Composition Semantics

- Performances and processes have the following execution cycle:
  - They fulfill their pre-conditions (such as waiting for inputs to become available etc.).
  - Signal readiness via channel r.
  - Wait for permission to execute from their environment via channel e.
  - Perform their internal activity.
  - The clock σ ticks to indicate overall completion (being no longer held up by maximal progress).
- Each performance is assigned and composed with a *scheduler* which dictates when it can run.

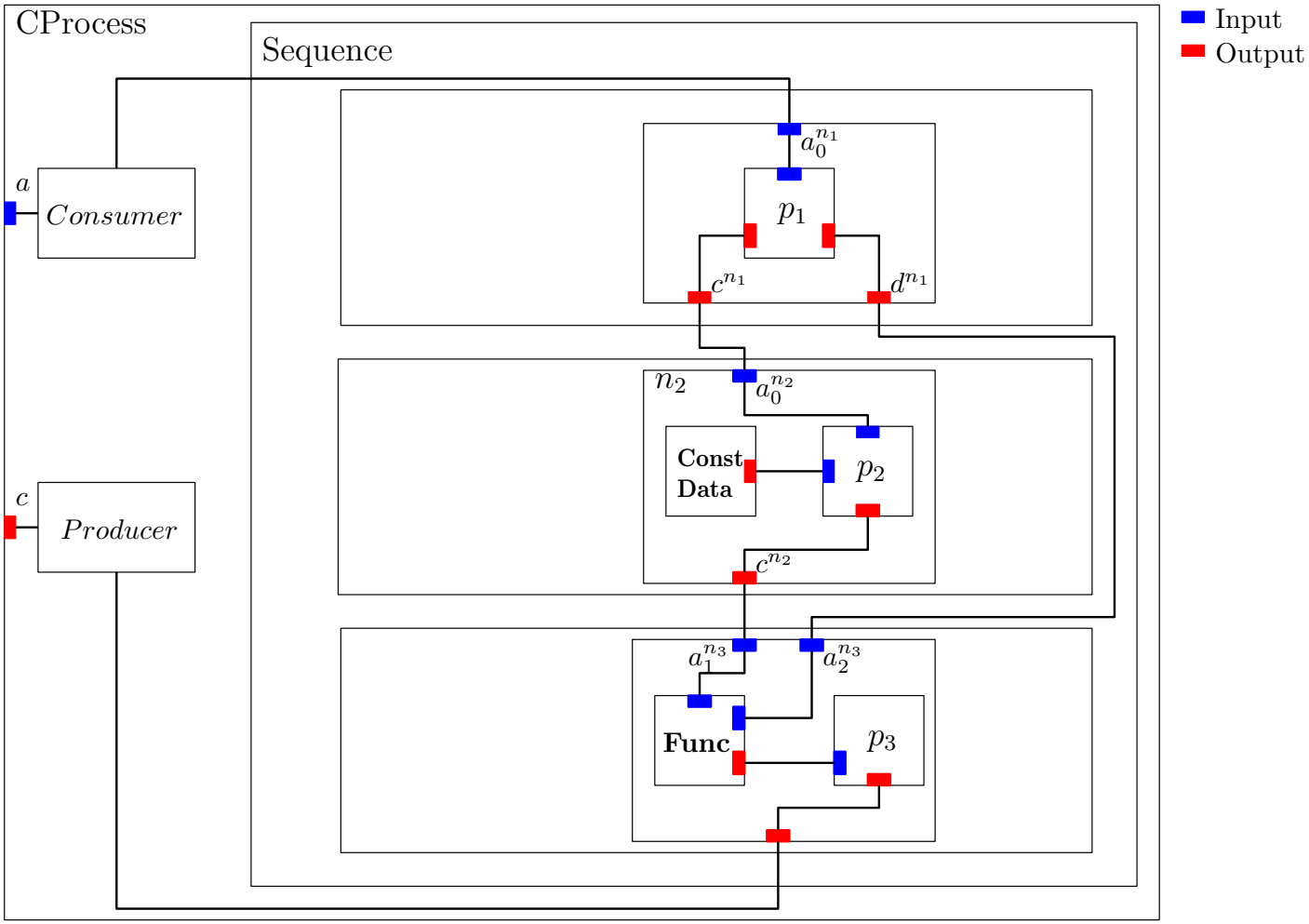
Semantics for Sequence scheduler

Base Case $\mu X.\underline{r^i.\overline{r}.e.\overline{e^i}}_{\{\sigma^m,\sigma^n\}}.\underline{\sigma^n}_{\sigma^m}\lfloor\underline{\overline{t}.\sigma^m}_{\sigma^n}.X\rfloor\sigma^m(X)$

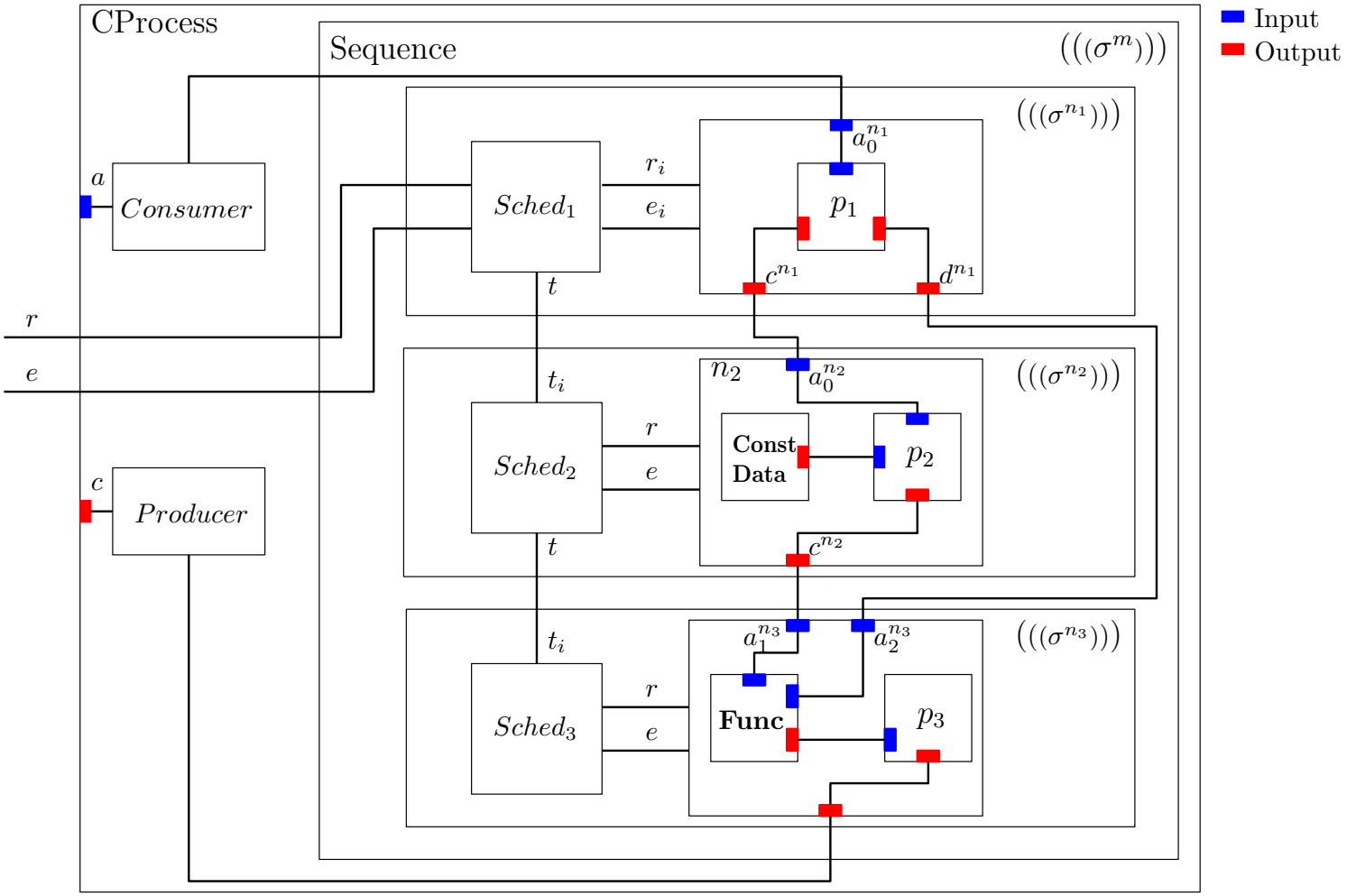
Inductive Case $\mu X.\underline{t^i}_{\sigma^n}.\underline{r^i.\overline{e^i}}_{\{\sigma^m,\sigma^n\}}.\underline{\sigma^n}_{\sigma^m}\lfloor\underline{\overline{t}.\sigma^m}_{\sigma^n}.X\rfloor\sigma^m(X)$

- r/e - readiness/permission to execute
- t - the token (owned by the performance currently executing)
- σ^n signals completion of each component
- σ^m signals completion of the whole process

Sequential Composition Example



Sequential Composition Example - with scheduling



Advantages of this model

- It is compositional;
- It is finite state (allows automata-based model-checking);
- It enables component replacement via temporal observation congruence;
- It is simple to implement in a functional programming language;
- It can be expanded to provide more features without changing the calculus.

Outline

Compensable Transactions

- Key to the notion of *long running transactions*, which can be composed using languages like BPEL.
- Allows rollback of all or part of a transaction according to some specific ordering rules (e.g. Sequences should be rolled back in reverse from point of failure)
- Compensation in Cashew-S should be compositional such that compensation is not undermined if a process evolves.

Compensable Transactions (cont.)

- Initial experiments show that Cashew-Nuts sufficiently expressive to represent compensation similar to that modelled in Sagas calculi and cCSP [Bruni et al., 2005].
- Three new syntactic elements should be sufficient for compensations:
  - **Transaction**, which defines a transaction block, where the components "compensate together".
  - **Compensate**, called $\div$ elsewhere, defines the performance which should be applied to compensate for another performance.
  - **Raise**, defines when an event (e.g. an exception) has occurred which needs to be compensated for.

Compensations for Cashew-Nuts

- Compensation, as in cCSP and Sagas calculi, is driven by three princaple signals
  - Completion ($\boxdot$), on success
  - Compensation ($\boxtimes$), when the process needs to compensate
  - Abnormal Termination ($\circledast$), when compensation has failed
- The former signal is already given in the form of a σ-tick
- The latter two can also be handled by clocks (though probably sans maximal progress)
- Each successful performance installs (via enabling) its compensation
- Compensations then run when the compensation clock ticks (order prescribed by workflow pattern).

Integration with WSMO

- Cashew-S needs to become more goal oriented.
- Currently just about executing operations of services in a particular order.
- OWL-S had no choreography - interaction with "services" was in reality merely the calling of operations.
- In Cashew-S a declarative language is needed which would enable description of this interaction in a goal-wise manner.
- Conformance between choreography and orchestration established by temporal observation congruence.

Implementation

- Two implementations of Cashew are planned
- One in Haskell (already underway), which is primarily for validation of the semantics and exploration.
- One in Lisp, for integration into the *Internet Reasoning Service* (one of the two most active implementations of WSMO being developed at KMi).

Outline

Conclusion

- We have looked at the current state of service composition, and which features are required.
- The Cashew project has thus far given a compositional operational semantics to orchestration, using the process calculus Cashew-Nuts.
- Cashew-Nuts needs some more work on the theoretical side (e.g. data-typing issues, algebraic theory).
- We are currently working on adding compensation transactions to orchestration.
- Cashew-S will be given a choreography language so that it can be integrated with WSMO/IRS.

References I


B. Norton, G. Lüttgen, and M. Mendler.
A compositional semantic theory for synchronous component-based design.
In *14th Intl. Conference on Concurreny Theory (CONCUR '03)*, number 2761 in LNCS. Springer-Verlag, 2003.


D. Roman, U. Keller, H. Lausen et al.
Web Service Modelling Ontology
In *Applied Ontology 1 (2005) 77–106*

References II


B. Norton, S. Foster, and A. Hughes.
A compositional operational semantics for OWL-S.
In *Proc. 2nd Intl. Workshop on Web Services and Formal Methods (WS-FM 2005)*, September 2005.


W. van der Aalst.
Pi Calculus Versus Petri Nets: Let Us Eat Humble Pie Rather Than Further Inflate the Pi Hype.
Business Process Trends, 2005.


R. Bruni, M. Butler, C. Ferreira, T. Hoare, H. Melgratti, and U. Montanari
Comparing two approaches to compensable flow composition.
Proc. of CONCUR 2005, the 16th International Conference on Concurrency Theory, 2005.

Thank you for listening

S.Foster@dcs.shef.ac.uk

http://www.dcs.shef.ac.uk/~simonf