Modelling Compensation with Timed Process Algebra

Simon Foster ⟨S.Foster@dcs.shef.ac.uk⟩

Department of Computer Science
University of Sheffield

11/06/2007

Outline

Compensation

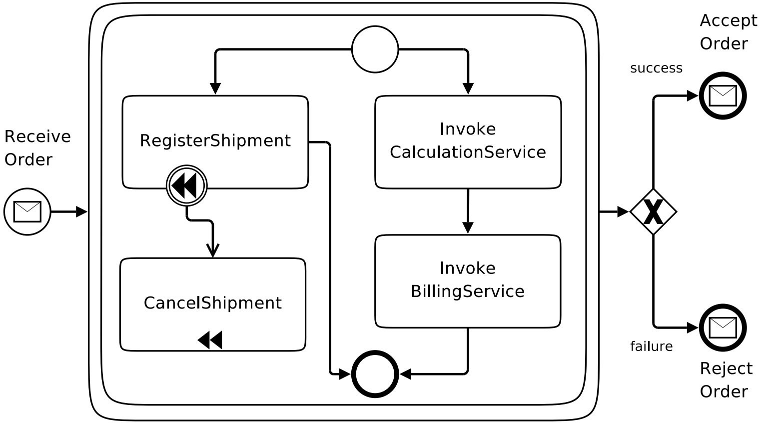
Cashew-S

Behavioural Semantics

What is compensation?

- A form of transaction support where you don't have ACID properties.
- A series of service interactions to try and undo half completed work when failure occurs.
- The compensation is run in the reverse direction to the normal forward flow.
- We split orchestration into discrete *transaction blocks*.
- Each component's "forward flow" is associated with compensation actions – "compensation flow".

Example compensable transaction



Compensable flow languages

- Our objective is a component-oriented compensable flow language, with a general approach to compensable patterns.
- Compensation can be modelled in a number of different ways:
  - Centralised with Interruption;
  - Distributed.
- Our framework should support all of these.
- Rather than introducing purpose specific constructs into a language we want to use more canonical process algebraic constructs.

Outline

Compensation

Cashew-S

Behavioural Semantics

Cashew-S

- An orchestration language.
- Originally based on OWL-S process model, though with several extra features.
- Uses workflow patterns to compose *performances*.
- Performances may be
  - Service interaction (Send, Receive etc.);
  - Expression Evaluation;
  - Workflow encapsulation;
  - Transactions.
- All performances are named and have inputs and outputs.
- p represents a performance name, w a workflow name.
- The compensable fragment of this language follows.

$$
\begin{array}{rcl}
\mathit{Transaction} & ::= & \textbf{Perform}\ p\ \textbf{Transaction}\ \mathit{CWorkflow} \\
\mathit{TransList} & ::= & \mathit{Transaction} \mid \mathit{TransList}; \mathit{Transaction} \\
\mathit{CWorkflow} & ::= & \textbf{Workflow}\ w\ (\mathit{Acceptors})\ (\mathit{Offerors})\ \mathit{CPattern} \\
\mathit{CPattern} & ::= & \textbf{Seq}\ (\mathit{CPerfList}) \quad \mid \textbf{Par}\ (\mathit{CPerfList}) \\
 & \mid & \textbf{Inter}\ (\mathit{CPerfList}) \mid \textbf{Conc}\ z\ z\ z\ (\mathit{TransList}) \\
 & \mid & \textbf{Choice}\ (\mathit{CPerfList}) \mid \textbf{Skip} \mid \textbf{Throw} \mid \textbf{Yield} \\
\mathit{CPerf} & ::= & \mathit{AtomicPerformance} \mid \mathit{Compensation} \\
 & \mid & \mathit{CWfPerf} \mid \mathit{Transaction} \\
\mathit{CWfPerf} & ::= & \textbf{Perform}\ p\ \mathit{CWorkflow} \\
\mathit{Compensation} & ::= & \mathit{CPerf} \div \mathit{Performance} \\
\mathit{CPerfList} & ::= & \mathit{CPerf} \\
 & \mid & \mathit{CPerfList}; \mathit{Connection} \\
 & \mid & \mathit{CPerfList}; \mathit{CPerf}
\end{array}
$$

Compensation in Cashew-S

- A transaction follows its forward flow as dictated by the workflow.
- When an exception is raised the flow switches direction and the compensations installed so far are run.
- Exceptions are not propogated beyond the transaction block and compensations cannot fail.

Outline

Compensation

Cashew-S

Behavioural Semantics

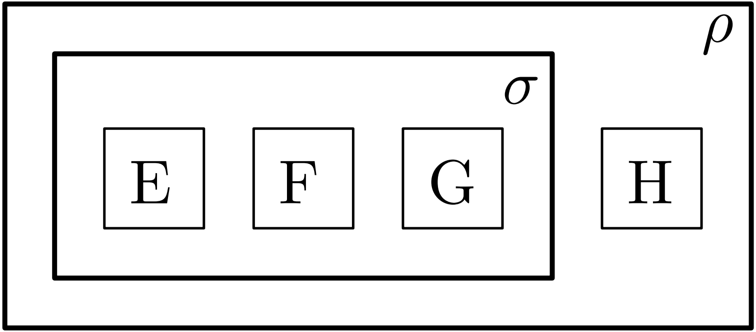
CaSiE

- CaSiE is a conservative extension of CCS.
- Adds **discrete time** in the form of multi-party synchronisation with **maximal progress**.
- The “clock” acts as a synchronisation point.
- Clocks are excluded rather than included – they implicitly exist and must be explicitly disabled.
- It also has a form of **interruption**, which allows any work to be preempted.
- Both time and interruption can be localised to a particular area of the system topology via **hiding**.

Synchronous Hierarchies

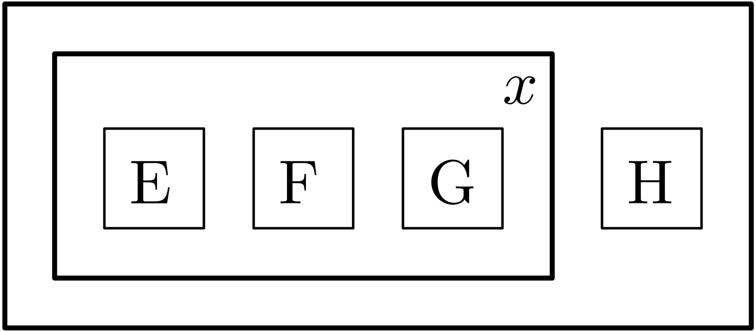
- Maximal progress and clock hiding allow the formation of *synchronous hierarchies*.
- In hiding a clock we define a synchronous block within which all agents synchronise (whilst unobservable outside).
- Since hiding forms converts a clock tick into a silent action which in turn prevents any clocks outside from ticking.
- Thus silent an implicit ordering on the blocks' behaviour results.

Synchronous Hiearchies



$$((E \mid F \mid G)/\sigma \mid H)/\rho$$

Localised Interruption



$$(E \mid F \mid G)/x \mid H$$

Equivalence theory

- Based on bisimulation with a congruence which abstracts over silent actions.
- Allows components to be check if they will behave the same in all contexts.
- Facilitates component drop-in within an orchestration.
- Axiomatised over the non-interruptible fragment (CaSE).

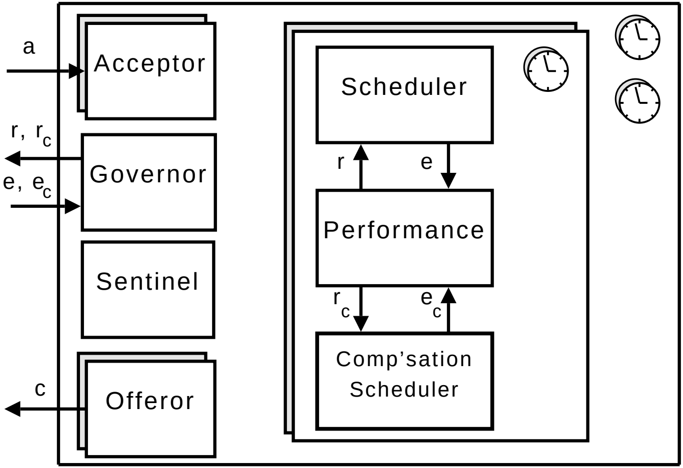
Operational Semantics

- We give Cashew-S a semantics in terms of CaSiE.
- Each performance in a workflow is given a semantics and associated with **Scheduler** which controls when it can run.
- Each workflow also has a **Governor** which communicates with the environment.
- Then in turn each workflow may be wrapped into a performance and itself becomes part of another workflow.

Transactions

- Each transaction has a **Sentinel**, which handles interruption by asking all sub-threads to yield and starting off the compensation process.
- The sentinel also handles the final “commit” by broadcasting a z signal to all compensation schedulers etc.

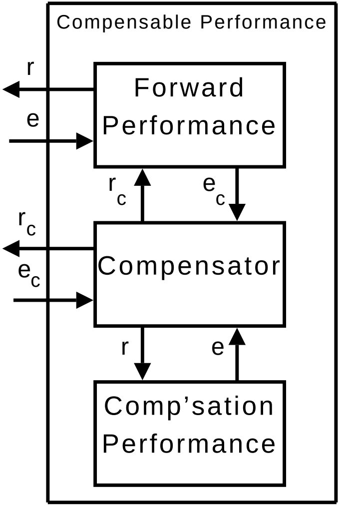
Agent architecture



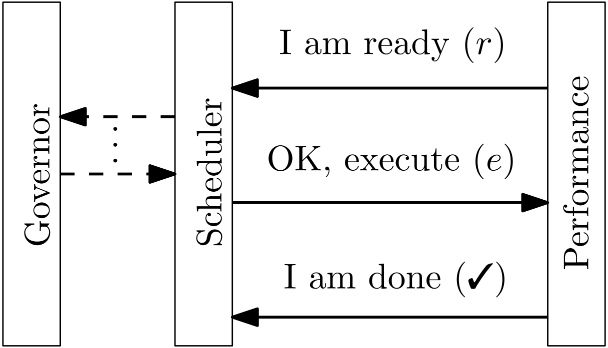
Compensation Pairs

- A compensation pair of the for $p_1 \div p_2$ is associated with a **Compensator** agent.
- The Compensator handles "installation" of the compensation performance when the forward performance finishes.
- Before successful completion, any compensation request are passed directly onto the forward flow.
- However afterward they are passed onto the compensation performance.

Compensator

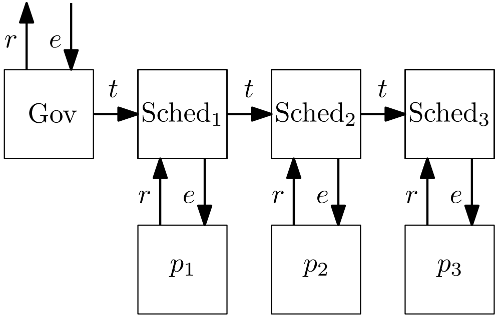


Negotiation Protocol

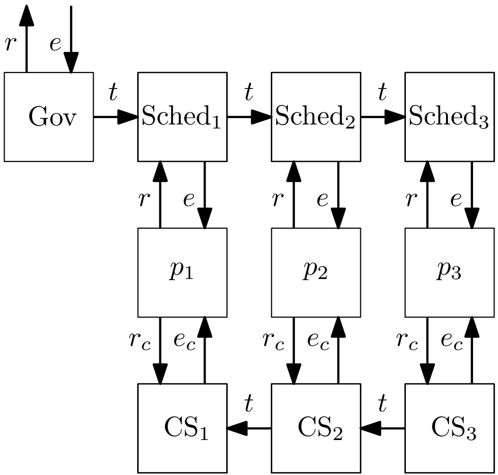


- Simplified somewhat.
- There are also some clocks involved to define the current phase of execution.

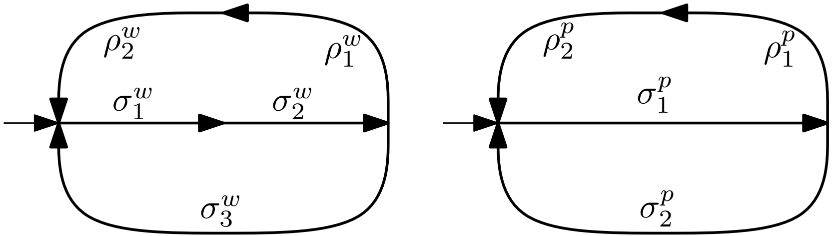
Sequential Workflow



Sequential Workflow with Compensation

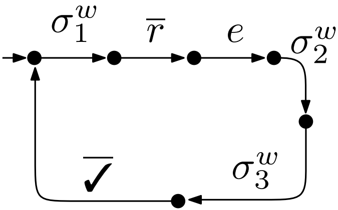


Workflow and Performance Phases

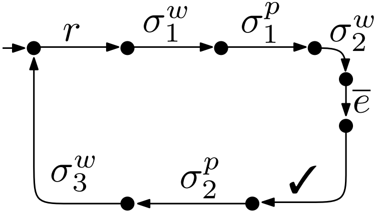


- Phases dictate the current macro state that a component is in, and allows agents to share the state.
- Each phase has an associated clock.
- σ clocks tick during the "normal" behaviour of a components, ρ clocks tick during exception handling.

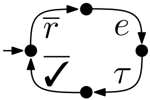
Par Scheduling



Governor

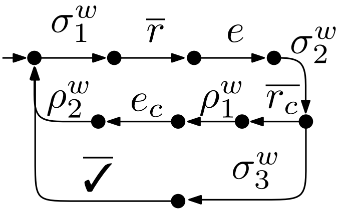


Scheduler

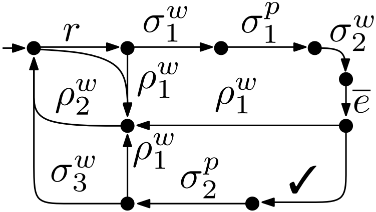


Performance

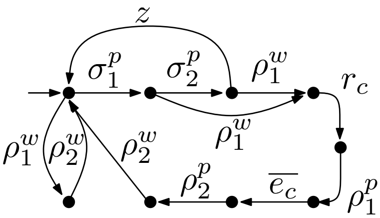
Par Scheduling + Compensation



Governor



Scheduler



Compensation Scheduler

Conclusion

- We have outlined a simplified model for compensation using timed process algebra.
- Our aim is to use this to give a semantics to different patterns of compensation, which can be used within Cashew-S.
- We are also working on an implementation of this for an orchestration engine in the functional programming language *Haskell*.