SCIENCE
UNIVERSITY OF COPENHAGEN

A Declarative Framework for Enterprise Systems

Christian Stefansen

Yale University, Feb. 3, 2006

SCIENCE
UNIVERSITY OF COPENHAGEN

Abstract

This talk sketches some recent research toward a declarative process-based framework for ERP systems. In particular, the talk describes the following two areas:

(1) A declarative language for compositional specification of contracts governing the exchange of resources. The language extends Eber and Peyton Jones's declarative language for specifying financial contracts to the exchange of money, goods and services amongst multiple parties, and it complements McCarthy's Resources, Events and Agents (REA) accounting model with a view-independent formal contract model that supports definition of user-defined contracts, automatic monitoring under execution, and user-definable analysis of their state before, during and after execution.

(2) A pi-calculus encoding of common workflow control patterns, which leads to a pi-calculus-based macro language for workflow specification. The encodings presented here demonstrate some of the strengths and weaknesses of pi-calculus for business process formalization vis a vis Petri nets and concrete languages such as BPEL and YAWL.

SCIENCE
UNIVERSITY OF COPENHAGEN

The NEXT Research Group

- NEXT generation Enterprise Resource Planning (ERP) systems
- ”The over-all goal of the project is to develop software technology and methods for development of the next generation of business management systems for companies. These systems are often referred to as ERP-systems (ERP: Enterprise Resource Planning) even though today they comprise much more than resource planning.”
- Fritz Henglein
  Henrik Reif Andersen
  Peter Sestoft
  Rune Møller Jensen
  Kasper Østerbye
  Peter Carstensen
  Yvonne Dittrich

SCIENCE
UNIVERSITY OF COPENHAGEN

What Is an Enterprise System?

- Usually centered around a general ledger, enterprise systems handle business by supporting:
  - Accounts payable/receivable
  - Inventory, assets, projects, resource allocation
  - Employees, workflows, tasks
  - Customer relations
  - Supply-chain management, logistics

SCIENCE
UNIVERSITY OF COPENHAGEN

The Goal

- ***To be able to write, automate, analyze and monitor business processes in a service-oriented architecture.***

- Current systems (SAP, PeopleSoft, Axapta, Siebel, Great Plains, Compiere) are not process-oriented:
  - ad hoc-integration between modules,
  - do not expose and share processes with the environment,
  - do not support mobility.

SCIENCE
UNIVERSITY OF COPENHAGEN

The Project

- Design a process-oriented programming platform.

- Processes, constraints, and rules
- Data model
- Reporting and monitoring

SCIENCE
UNIVERSITY OF COPENHAGEN

Compositional Commercial Contracts

Jesper Andersen Ebbe Elsborg

Fritz Henglein Jakob Grue Simonsen

Christian Stefansen

SCIENCE
UNIVERSITY OF COPENHAGEN

Parts of a Contract

- Definitions
  (define parties, nature and quality of exchanged resources, legal context, exception handlers etc.)
- Temporal/logic structure

- **The good news:** We can express temporal properties formally.
- **The bad news:** Probably cannot eliminate the need for lawyers altogether!

SCIENCE
UNIVERSITY OF COPENHAGEN

Paper-Based Contract-Handling is Costly

- No formal representation results in:
  - Manual handling in auxiliary systems
  - Ad hoc deadline management
  - Ambiguous semantics
  - Time-consuming valuation
  - Cannot report on future events
  - No coordination with production planning or supply-chain management
  - Missed opportunities (call options etc.)
- Potential benefits of formal representation:
  - Alleviate the problems above
  - Several consistency checks at time of writing contract
  - (Semi-)automated contract analysis (incl. valuation) (DSL programs are in a sense ”intelligent data”)

SCIENCE
UNIVERSITY OF COPENHAGEN

We Analyzed 15 Full-Length Contracts

- Data model
- Structure

Agreement to Sell Goods	Sale with Installment Payment
General Contract	Agreement to Sell
Balloon Note	Contractor Agreement
Legal Services Agreement	The Danish Trade Law
Website Development Contract	Lease Contract
Loan and Security Agreement	License Agreement
Operating Agreement	Supply Agreement
Manufacturing Agreement	

SCIENCE
UNIVERSITY OF COPENHAGEN

Example: Agreement to Provide Legal Services

- **Section 1.** The attorney shall provide, on a non-exclusive basis, legal services up to (n) hours per month, and furthermore provide services in excess of (n) hours upon agreement.
- **Section 2.** In consideration hereof, the company shall pay a monthly fee of (amount in dollars) before the 8th day of the following month and (rate) per hour for any services in excess of (n) hours 40 days after the receival of an invoice.
- **Section 3.** This contract is valid 1/1-12/31, 2008.

SCIENCE
UNIVERSITY OF COPENHAGEN

Primitives in Contracts

- Data
  - Agents
  - Resources (goods, services, rights)
  - Commitments/Events
  - Time
- Structure
  - Sequential execution
  - Concurrent execution
  - Repeated execution
  - Alternative execution (choice between subcontracts)

SCIENCE
UNIVERSITY OF COPENHAGEN

Syntax

Success/Failure
: The succeeded/failed contract with no commitments.

transmit$(A_1, A_2, R, T|P).c$
: The commitment of agent A_1 to transmit resource R to agent A_2 at time T subject to predicate P (afterwards do contract c).

$c_1; c_1$
: A sequence of two contracts. The first contract must be reduced to Success before the second can begin.

$c_1 \parallel c_2$
: Parallel, independent execution of two contracts.

$c_1 + c_2$
: (Non-deterministic) choice between two contracts.

$f(\vec{a})$
: Expansion to body of contract f with arguments $\vec{a}$.

letrec $f_i[\vec{X_i}] = c_i$ **in** c
: Contract c with named contracts f_i with formal arguments X_i bound to c_i.

SCIENCE
UNIVERSITY OF COPENHAGEN

Example: Legal Agreement Code

```
letrec
extra (att, com, invoice, pay) =
   ( Success
   + transmit (att, com, invoice, T2).
     transmit (com, att, pay, T3 | T3 <= T2 + 45d))

legal (att, com, fee, invoice, pay, n, m, end) =
    transmit (att, com, H, T | n < T and T <= m).
        (   extra (att, com, invoice, pay)
        ||  transmit (com att, fee, T | T <= m + 8d)
        ||  ( legal (att, com, fee, invoice, pay, m, min(m + 30d,end)
            + transmit (att, com, end, T | end <= T)))
in
  legal ("Attorney","Company",10000,invoice,pay,0,30,360)
```

SCIENCE
UNIVERSITY OF COPENHAGEN

When Is a Contract Satisfied?

- Contracts denote sets of finite traces. A trace is a finite sequence of events:

  s ::= <> | transmit(*a1,a2,r,t*) *s*

- So contracts classify a given trace as performing or nonperforming. Formally:

$$\mathcal{C}[\![c]\!]^{\mathcal{D}[\![\mathrm{D}]\!]^{\delta};\delta\oplus\delta'} = \{s \mid \delta' \vdash^{\delta}_{\mathrm{D}} s : c\}$$

SCIENCE
UNIVERSITY OF COPENHAGEN

The Satisfaction Relation

$$\frac{\delta \oplus \delta'' \models P \quad \delta'' \vdash^{\delta}_{\mathrm{D}} s : c \qquad (\delta'' = \delta' \oplus \{\boldsymbol{X} \mapsto \boldsymbol{v}\})}{\delta' \vdash^{\delta}_{\mathrm{D}} \mathrm{transmit}(\boldsymbol{v})\, s : \mathrm{transmit}(\boldsymbol{X}|P).\, c}$$

$$\delta' \vdash^{\delta}_{\mathrm{D}} \langle\rangle : \mathrm{Success}$$

$$\frac{\delta' \vdash^{\delta}_{\mathrm{D}} s_1 : c_1 \quad \delta' \vdash^{\delta}_{\mathrm{D}} s_2 : c_2 \quad (s_1, s_2) \rightsquigarrow s}{\delta' \vdash^{\delta}_{\mathrm{D}} s : c_1 \parallel c_2}$$

- How about *Failure*? (Exercise for the audience)
- Note that the base language is orthogonal to this. Just plug in your own.

SCIENCE
UNIVERSITY OF COPENHAGEN

From the Satisfaction Relation to Denotational Semantics

- Denotational semantics, e.g.:

$$\mathcal{C}[\![c_1 + c_2]\!]^{\gamma;\delta} = \mathcal{C}[\![c_1]\!]^{\gamma;\delta} \cup \mathcal{C}[\![c_2]\!]^{\gamma;\delta}$$

- Theorem: Denotational characterization of satisfaction

$$\mathcal{C}[\![c]\!]^{\mathcal{D}[\![\mathrm{D}]\!]^{\delta};\delta\oplus\delta'} = \{s \mid \delta' \vdash^{\delta}_{\mathrm{D}} s : c\}$$

SCIENCE
UNIVERSITY OF COPENHAGEN

Toward Contract Execution

- We need a representation of residual contracts in order to monitor execution.
- For a trace set ***S*** and an event ***e***, define the residuation function as:

$$e\backslash S = \{s' \mid \exists s \in S : es' = s\}$$

SCIENCE
UNIVERSITY OF COPENHAGEN

Residual Contracts Should Be Expressible

- Let *Sc* denote the trace set of contract *c*.
- Not immediately clear if *e\Sc* is denotable by a contract (syntactical expression) *c'*
- But this is clearly a normative property!

SCIENCE
UNIVERSITY OF COPENHAGEN

Not All Residuals Are Representable

$$\text{letrec}\, f(N) = (\text{transmit}(a_1, a_2, r, T \mid T \leq N) \parallel f(N+1)) \text{ in } f(0)$$

Assume event transmit(*a1, a2, r, 0*) occurs.

$$\text{transmit}(a_1, a_2, r, T_0 | T_0 \leq 0) \parallel \cdots \parallel \text{transmit}(a_1, a_2, r, T_i | T_i \leq i) \parallel \cdots$$

We therefore introduce *guardedness*, a sufficient condition to ensure that all contracts have a syntactic representation of the residual contract under any event.

SCIENCE
UNIVERSITY OF COPENHAGEN

Guarded Contracts

- Intuitively, a contract is guarded if (mutual) recursive calls are prefixed by a transmit.

$$\mathrm{D} \vdash \text{Success guarded} \qquad \mathrm{D} \vdash \text{Failure guarded}$$

$$\mathrm{D} \vdash \text{transmit}(\boldsymbol{X} \mid P).\, c \text{ guarded} \qquad \frac{\mathrm{D} \vdash c \text{ guarded} \quad (f(\boldsymbol{X}) = c) \in \mathrm{D}}{\mathrm{D} \vdash f(\boldsymbol{a}) \text{ guarded}}$$

$$\frac{\mathrm{D} \vdash c \text{ guarded} \quad \mathrm{D} \vdash c' \text{ guarded}}{\mathrm{D} \vdash c + c' \text{ guarded}} \qquad \frac{\mathrm{D} \vdash c \text{ guarded} \quad \mathrm{D} \vdash c' \text{ guarded}}{\mathrm{D} \vdash c \parallel c' \text{ guarded}}$$

$$\frac{\mathrm{D} \vdash c \text{ nullable} \quad \mathrm{D} \vdash c \text{ guarded} \quad \mathrm{D} \vdash c' \text{ guarded}}{\mathrm{D} \vdash c; c' \text{ guarded}} \qquad \frac{\mathrm{D} \nvdash c \text{ nullable} \quad \mathrm{D} \vdash c \text{ guarded}}{\mathrm{D} \vdash c; c' \text{ guarded}}$$

SCIENCE
UNIVERSITY OF COPENHAGEN

Several Operational Semantics

- Now that guarded contracts are representable, we can attempt defining an operational semantics to govern contract rewriting. We defined three inter-related semantics:
- Non-deterministic eager matching
- Deterministic eager matching with explicit routing
- Deterministic reduction by delayed matching

SCIENCE
UNIVERSITY OF COPENHAGEN

Deterministic Reduction with Delayed Matching

- Consider some of the reduction rules:

$$\frac{\delta \oplus \{\boldsymbol{X} \mapsto \boldsymbol{v}\} \models P \quad (\boldsymbol{v} = \mathcal{Q}[\![\boldsymbol{a}]\!]^{\delta})}{\mathrm{D}, \delta \vdash_D \mathrm{transmit}(\boldsymbol{X}|P).\, c \overset{\mathrm{transmit}(\boldsymbol{v})}{\longrightarrow} c[\boldsymbol{v}/\boldsymbol{X}]}$$

$$\frac{\mathrm{D}, \delta \vdash_D c \overset{e}{\longrightarrow} d \quad \mathrm{D}, \delta \vdash_D c' \overset{e}{\longrightarrow} d'}{\mathrm{D}, \delta \vdash_D c + c' \overset{e}{\longrightarrow} d + d'}$$

$$\frac{\mathrm{D}, \delta \vdash_D c \overset{e}{\longrightarrow} d \quad \mathrm{D}, \delta \vdash_D c' \overset{e}{\longrightarrow} d'}{\mathrm{D}, \delta \vdash_D c \parallel c' \overset{e}{\longrightarrow} c \parallel d' + d \parallel c'}$$

SCIENCE
UNIVERSITY OF COPENHAGEN

Guardedness Ensures Safe Residuation

- Guarded Subject Reduction:

For any c, c', δ, e *and* D*: if* $\mathrm{D}, \delta \vdash_D c \overset{e}{\longrightarrow} c'$ *then* $\mathrm{D}, \delta \models e \backslash c = c'$.
For all c, δ *and guarded* D*, there exists a unique* c' *such that* $\mathrm{D}, \delta \vdash_D c \overset{e}{\longrightarrow} c'$*;* $\mathrm{D} \vdash c'$ guarded.

- Now we reap the benefits of compositionality: all residual contracts have syntactical representations and can be submitted to analysis etc.

SCIENCE
UNIVERSITY OF COPENHAGEN

Example Reduction

```
transmit (att, com, H, T | 0 < T and T <= 30).
(   transmit (com, att, fee, T | T <= 30 + 8d)
||  ( legal (..., 30, min(30 + 30d,60), 60)
    + transmit (att, com, end, T | 60 <= T)))
```

Services rendered first month:

$$\xrightarrow{(att,com,h1,20)}$$

```
(   transmit (com, att, fee, T | T <= 30 + 8d)
||  ( legal (..., 30, min(30 + 30d,60), 60)
    + transmit (att, com, end, T | 60 <= T)))
```

Services rendered second month:

$$\xrightarrow{(att,com,h2,37)}$$

SCIENCE
UNIVERSITY OF COPENHAGEN

Example Reduction - Step 2

```
(   transmit (com, att, fee, T | T <= 30 + 8d)
||  ( legal (..., 30, min(30 + 30d,60), 60)
    + transmit (att, com, end, T | 60 <= T)))
```

Services rendered second month:

$$\xrightarrow{(att,com,h2,37)}$$

```
  (   Failure
  ||  ( legal (..., 30, min(30 + 30d,60), 60)
      + transmit (att, com, end, T | 60 <= T)))
+
  (   transmit (com, att, fee, T | T <= 30 + 8d)
  ||  (   (   transmit (com, att, fee, T | T <= 60 + 8d)
          ||  ( legal (..., 60, min(60 + 30d,60), 60)
              + transmit (att, com, end, T | 60 <= T)))
      + Failure))
```

SCIENCE
UNIVERSITY OF COPENHAGEN

Contract Analysis

- Compositional and thus available at runtime too!
- Wish to answer questions ranging from simple (what is my current todo list?) to complex (what is the current value of the contract based on my stochastic model?)
- Particularly important analysis: *Failure*, i.e. does the contract have no satisfying traces.

$$\frac{\forall \delta', \forall t' \geq t : (\delta \oplus \delta' \oplus T \mapsto t' \models \neg P)}{\mathrm{D}, \delta, t \vdash \mathrm{transmit}(\boldsymbol{X}T \mid P).\, c \text{ failed}}$$

SCIENCE
UNIVERSITY OF COPENHAGEN

Expressing Workflows in a Process Calculus

(or finding a good representation if one does not already exist)

Work in progress...

SCIENCE
UNIVERSITY OF COPENHAGEN

What Are Workflows?

- *"A business process is a collection of interrelated work tasks, initiated in response to an event, that achieves a specific result for the customer of the process."*
  -- Sharp and McDermott
- A tendency to consider workflows a subset of business processes
- But I have not yet found (or needed) a workable, rigorous distinction.

SCIENCE
UNIVERSITY OF COPENHAGEN

Why Formal Workflows?

- Ad hoc workflow handling:
  - Error-prone (no guidance)
  - Significant training needed
  - Lends itself badly to analysis and change
- Formal representation of workflows:
  - Alleviate the above
  - Execution model (computers can execute workflows)
  - Static and runtime consistency checks
  - Lends itself well to oursourcing, service-oriented architecture, and partial automation

SCIENCE
UNIVERSITY OF COPENHAGEN

Why Formal Workflows?

- Users
  - can easily adhere to established best practice
  - know what tasks kan be dealt with now/later
  - receive help to delegate tasks appropriately
  - need only local knowledge about the tasks they solve (as opposed to global knowledge about the entire workflow)
- Designers/planners
  - can more easily map out and change processes
  - can introduce structure along the way (ad hoc)
  - can perform formal analysis on workflows
  - can partially automate outsourcing etc.
- Controllers
  - gain finer registration of resource consumption (e.g. time) and thus costs (get Activity-Based Costing for free)
  - can carry out performance analysis more easily

SCIENCE
UNIVERSITY OF COPENHAGEN

..but this is Taylor's Scientific Management all over again!

- Workflows can be as freeform or as rigid as one designs them.
- The simplest workflow is just a completely unstructured task list.
- Workflow can be adaptive, starting with the completely unstructured workflow first and imposing structured constraints over time.

SCIENCE
UNIVERSITY OF COPENHAGEN

Do Workflows Have A Semantics?

- No widely accepted semantics so far.
- Studies by Aalst et al. proposed:
  - 20 control-flow patterns (parallel, sequence, choice, repetition etc.)
  - 39 data patterns (scope, call-by regime, parameters etc.)
  - 43 resource patterns (delegation) (implicit, queue, role, case, quality check)

Today's topic

- The patterns are:
  - imprecise (no clear semantics)
  - overlapping (patterns overlap in non-obvious ways)
  - too inclusive (maybe "goto"-type patterns should be eliminated)
  - non-exhaustive (easy to think of more patterns, but a proper language should be defined instead)

SCIENCE
UNIVERSITY OF COPENHAGEN

The 20 Control-Flow Patterns

Basic Control Patterns

1 Sequence
2 Parallel Split
3 Synchronization
4 Exclusive Choice
5 Simple Merge

Advanced Branching and Synchronization Patterns

6 Multiple Choice
7 Synchronizing Merge
8 Multiple Merge
8a N-out-of-M Merge ***(new)***
9 Discriminator
9a N-out-of-M Join

Patterns Involving Multiple Instances

12 MI without synchronization
13 MI with a priori known design time knowledge
14 MI with a priori known runtime knowledge
15 MI with no a priori runtime knowledge

Structural Patterns

10 Arbitrary Cycles
11 Implicit Termination

Cancellation Patterns

19 Cancel Activity
20 Cancel Case

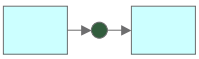
State-Based Patterns

16 Deferred Choice
16a Deferred Multiple Choice ***(new)***
17 Interleaved Parallel Routing
18 Milestone

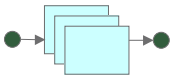
SCIENCE
UNIVERSITY OF COPENHAGEN

Some Patterns

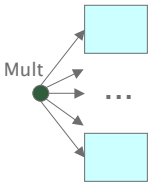
Pattern 1: Sequence
Execute activities in sequence.



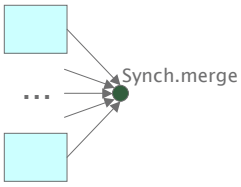
Pattern 13: Multiple Instances [...]
Generate many instances of one activity [...] with synchronization.



Pattern 6: Multiple Choice
Choose several execution paths from many alternatives.



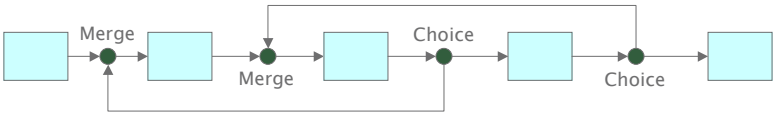
Pattern 7: Synchronizing Merge
Merge many execution paths. Synchronize if many paths are taken. [...] Merge if only one [...] path is taken.



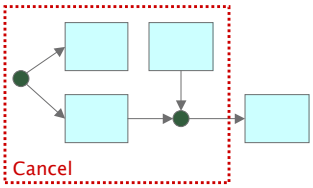
SCIENCE
UNIVERSITY OF COPENHAGEN

More Patterns

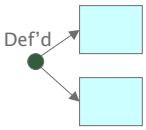
Pattern 10: Arbitrary Cycles
Execute workflow graph without any structural restriction on loops.



Pattern 20: Cancel Case
Cancel (disable) the process)



Pattern 16: Deferred Choice
Execute one of two alternative threads. The choice [...] should be implicit.



SCIENCE
UNIVERSITY OF COPENHAGEN

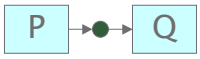
Calculus of Communicating Systems

- CCS syntax:

$$P \quad ::= \quad \mathbf{0} \mid \tau.P \mid a?x.P \mid a!x.P \mid P + P \mid (P \mid P) \mid \mathsf{new}\ a\ P \mid a?*x.P$$

Atomic tasks are simply written as *a,b,c* etc.

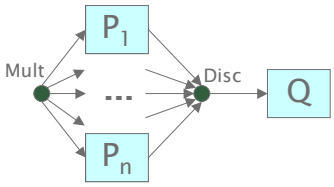
- **Pattern 1: Sequence**
  - a.P – not general enough
  - P.Q – not syntactic
  - Require processes to signal termination explicitly on a designated channel, e.g. *ok*
  - new *go* (P [*go*/*ok*] | *go*?.Q)



SCIENCE
UNIVERSITY OF COPENHAGEN

Example Encoding

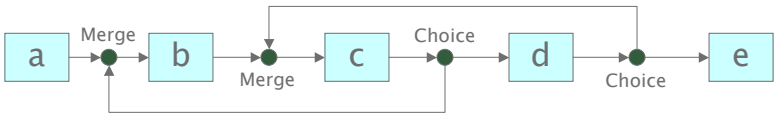
- **Pattern 7/9: Multiple Choice/Discriminator**
  - ($\tau.P_1$ + τ.skip!) | … | ($\tau.P_n$ + τ.skip!) | ok?.(Q | skip?*.**0** | ok?*.**0**)



SCIENCE
UNIVERSITY OF COPENHAGEN

Example Encoding

- **Pattern 10: Arbitrary Cycles**
  - new go_b, go_c
    ($a.go_b! \mid go_b?.*.b.go_c! \mid$
    $go_c?.*.c.(go_b! + d.(go_c! + e))$



- What if Merge was Sync. Merge?

SCIENCE
UNIVERSITY OF COPENHAGEN

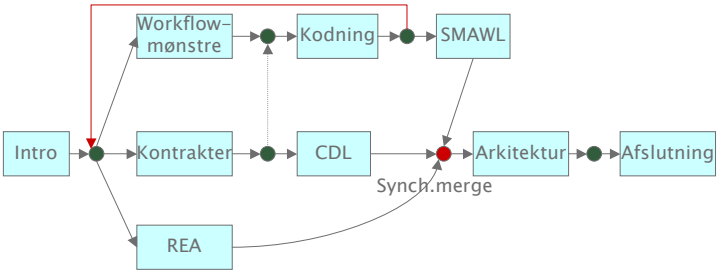
Observations on the patterns

- **Split and join are separate / Free structure**
  Patterns have graph-type structure (YAWL, Petri nets, WFDL) as opposed to block structure (XLANG, parts of BPEL)
- **Deferred vs. explicit choice**
  Branching time semantics may be needed
- **Many split and join variants seem better expressed by simple data-language**
- **Cancellation**
  Neither Petri nets nor pi-calculus handle this very well. LOTOS and CSP have operators for this.
- ***Multiple Instances* needed**
  Generativity essential

SCIENCE
UNIVERSITY OF COPENHAGEN

Synch. Merge Has Non-Local Semantics

- Workarounds
  1. Integrate with split-end pattern
     (thus loosing free-form property)
  2. Decide runtime
     (thus loosing some static checks)
  3. Pass around meta-information from previous split(s)
     (thus complicating distribution and introducing
     complexity, also insufficient in degenerate cases)



SCIENCE
UNIVERSITY OF COPENHAGEN

More Issues

- Semantics of *Discriminator* is unclear
  (but can be well-defined)
- Can tasks be cancelled?
  (sure, tasks can be processes too!)
- *Multiple Instances* combine several other patterns
  (split, join, and repetition)
- *Synch*, *Merge* and *Sequence* overlap somewhat
  subtly

SCIENCE
UNIVERSITY OF COPENHAGEN

A Formal Semantics for the Patterns?

- More precise benchmarking of workflow languages
- Should be model independent (not Petri net, π, EPC)
- Should split and join be separated or not?
  (and how about *Multiple Instances*?)
- Should all patterns be covered? Directly or by data-manipulation language?
- How to compare expressiveness? Felleisen-style?

SCIENCE
UNIVERSITY OF COPENHAGEN

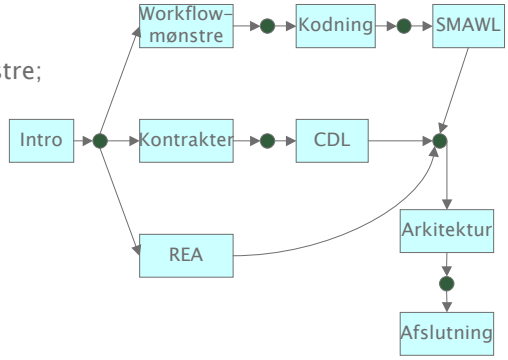
SMAWL – a SMAll Workflow Language

- Preliminary language design exercise
- Design criteria:
  - Cover all 20 patterns (and more!)
  - Minimize need for synchronization primitives
  - Retain strong link to pi/CCS
  - Independent of data patterns
    (Allows orthogonal data manipulation language to be plugged in.)
- Method
  - Collect patterns in a few, general constructions
  - Describe a source-code transformation to pi/CCS

SCIENCE
UNIVERSITY OF COPENHAGEN

A Workflow in SMAWL

```
workflow Q =
    Intro;
    choose any {
        ⇒ Workflowmønstre;
          Kodning;
          SMAWL
        ⇒ Kontrakter;
          CDL
        ⇒ REA
    };
    Arkitektur;
    Afslutning
end
```



SCIENCE
UNIVERSITY OF COPENHAGEN

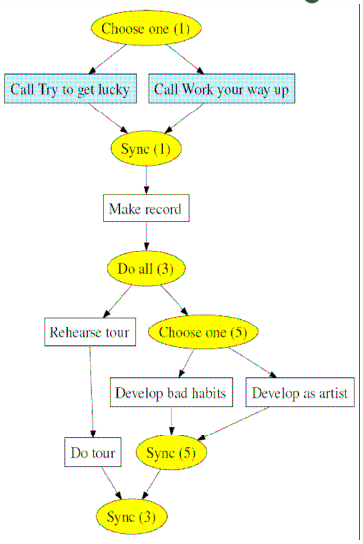
SMAWL Syntax

$$
\begin{array}{rcl}
W & ::= & \textbf{workflow}\ w = P\ \textbf{end} \\
D & ::= & \textbf{fun}\ f = P\ \textbf{end}\ D \mid \textbf{newlock}\ (l,u)\ D \\
 & \mid & \textbf{milestone}(ison, isoff, set, clear)\ D \mid \epsilon \\
P & ::= & activity \mid \textbf{send}(f) \mid \textbf{receive}(f) \mid \textbf{call}(f) \mid P;\ P \mid \textbf{lock}(l,u)\{P\} \\
 & \mid & \textbf{choose any (wait for k)}\{PP\ \textbf{merge}(n)\ P\} \mid \textbf{choose one}\{PP\} \mid \textbf{cancel}\ \{P\} \\
 & \mid & \textbf{do all (wait for k)}\{PP\ \textbf{merge}(n)\ P\} \mid \textbf{multi}(n)\{P\} \\
PP & ::= & \Rightarrow \rho\ P\ PP \mid P\ PP \mid \Rightarrow P
\end{array}
$$

SCIENCE
UNIVERSITY OF COPENHAGEN

Britney Wants to Sing

```
workflow Become a recording star =
    chooseone {
        ⇒ call (Work your way up)
        ⇒ call (Try to get lucky)
    };
    Make record;
    doall {
        ⇒ chooseone {
            ⇒ Develop as an artist
            ⇒ Develop bad habits
            }
        ⇒ Rehearse tour;
            Do tour
    }
end
```



SCIENCE
UNIVERSITY OF COPENHAGEN

Transforming SMAWL to CCS

- Sequence

$$\mathcal{T}[\![P;Q]\!] = \lambda ok.\text{let } ok' \Leftarrow \nu() \text{ in } \mathcal{T}[\![P]\!]ok' \mid ok'?.\mathcal{T}[\![Q]\!]ok$$

- Multiple Instances

$$\mathcal{T}[\![\textbf{multi}(n)\ \{P\}]\!] = \lambda ok.\text{let } create \Leftarrow \nu() \text{ in let } ok' \Leftarrow \nu() \text{ in}$$
$$\underbrace{create!.\cdots.create!}_{n}.\underbrace{ok'?.\cdots.ok'?}_{n}.ok! \mid create?*.\mathcal{T}[\![P]\!]ok'$$

SCIENCE
UNIVERSITY OF COPENHAGEN

Future Research on Workflows

- Set up success criteria for process model and data-manipulation language
- Formal semantics for workflows
- Consider distributed workflows
- Find/adapt/design suitable process calculus
- Implement a workflow language

SCIENCE
UNIVERSITY OF COPENHAGEN

A Process-Oriented System Architecture

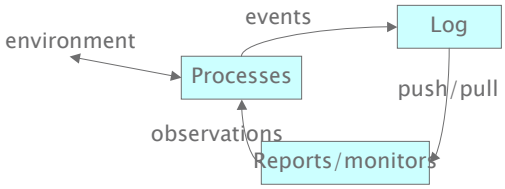
(Highly speculative)

SCIENCE
UNIVERSITY OF COPENHAGEN

Future: A Process-Oriented Framework

- Process language
- Base (data) language
- Service patterns
- Auto. GUI-generation

- Data model
- Process mining
- Inter-operability (ontology)



- Runtime verification
- Incrementalization of reports
- Stream processing
- Dynamic operator placement in overlay networks

SCIENCE
UNIVERSITY OF COPENHAGEN

Ongoing NEXT Projects

- Finite Differencing for Realtime Reporting
- REA Bookkeeping
  (Double-entry bookkeeping is not the only option)
- Plan X: Value-Based Programming

SCIENCE
UNIVERSITY OF COPENHAGEN

Related Work

- **Concurrency models**
  - Process calculi (CCS, pi, CSP, Bigraphs)
  - Petri nets
- **BPEL – Business Process Execution Language**
- **YAWL – Yet Another Workflow Language**
  Petri net-based workflow tool. Ostensibly covers all 20 control-flow patterns.
- **WS-CDL - Choreography Description Language**
  In particular, see Kohei Honda's recent stuff on behavioral type systems.

SCIENCE
UNIVERSITY OF COPENHAGEN

Related Work

- *Transactors: a programming model for maintaining globally consistent distributed state in unreliable environments*
  John Field and Carlos Varela, POPL '05
- *PiDuce – a project for experimenting with web services technologies*
  Samuele Carpineti, Cosimo Laneve, Luca Padovani, Jan. 2006
- *Compositional Contracts*
  Peyton Jones, Eber, Seward, 2001
- Also see: **http://www.process-modelling-group.org/**

SCIENCE
UNIVERSITY OF COPENHAGEN

Q&A and thanks!

- Thank you for hosting me!
- Questions?

- More info on:

  http://www.stefansen.dk

  http://www.it.edu/next/