Reliable Workflow in a Distributed Environment

William Cook, Jayadev Misra,
David Kitchin, Adrian Quark,
Andrew Matsuoka, John Thywissen

Department of Computer Science
University of Texas at Austin

http://orc.csres.utexas.edu

Orc Reliable Workflow HCSS Project

- Orc
  - Structured Concurrent Programming
  - Workflow/Internet Scripting Language
  - Ongoing project for last 5 years
- HCSS Project Topics
  - Time-based Semantics
  - **Implementation (Demo)**
  - Simulation/Logical Time
  - **Secure/Adaptive Workflow**
  - Data and Transactions
- (Present research directions)

Internet Scripting Example

- Contact two airlines simultaneously for price quotes
- Buy a ticket if the quote is at most $300
- Buy the cheapest ticket if both quotes are above $300
- Buy a ticket if the other airline does not give a timely quote
- Notify client if neither airline provides a timely quote

Orchestrating Components (services)

Acquire data from services
Calculate with these data
Invoke yet other services with the results

Additionally
Invoke multiple services simultaneously for failure tolerance
Repeatedly poll a service
Ask a service to notify the user when it acquires the appropriate data
Download a service and invoke it locally
Have a service call another service on behalf of the user
...

Structured Concurrent Programming

- Structured Sequential Programming: Dijkstra circa 1968
  Component Integration in a sequential world.

- Structured Concurrent Programming:
  Component Integration in a concurrent world.

  This is a significant challenge that needs focus from community

Orc, an Orchestration Theory

- Site: Basic service or component
- Concurrency combinators for integrating sites
- Theory includes nothing other than the combinators

No notion of data type, thread, process, channel, synchronization, parallelism $\cdots$

New concepts are programmed using the combinators

Examples of Sites

- $+ \; - \; * \; \&\& \; || \; < \; =$ ...
- `println`, `random`, `Prompt`, `Email` ...
- `Ref`, `Semaphore`, `Channel`, `Database` ...
- `Timer`
- External Services: Google Search, MySpace, CNN, ...
- Any Java Class instance
- Sites that create sites: `MakeSemaphore`, `MakeChannel` ...
- Humans

  ...

Sites

- A site is called like a procedure with parameters.
- Site returns at most one value.
- The value is published.

Site calls are strict.

Structure of Orc Expression

- Simple: just a site call, $CNN(d)$
  Publishes the value returned by the site

- Composition of two Orc expressions:

do f and g in parallel	$f \mid g$	Symmetric composition
for all x from f do g	$f >x> g$	Sequential composition
for some x from g do f	$f <x< g$	Pruning

Structure of Orc Expression

- Simple: just a site call, $CNN(d)$
  Publishes the value returned by the site

- Composition of two Orc expressions:

do f and g in parallel	$f \mid g$	Symmetric composition
for all x from f do g	$f >x> g$	Sequential composition
for some x from g do f	$f <x< g$	Pruning

Structure of Orc Expression

- Simple: just a site call, $CNN(d)$
  Publishes the value returned by the site

- Composition of two Orc expressions:

do f and g in parallel	$f \mid g$	Symmetric composition
for all x from f do g	$f > x > g$	Sequential composition
for some x from g do f	$f < x < g$	Pruning

Structure of Orc Expression

- Simple: just a site call, $CNN(d)$
  Publishes the value returned by the site

- Composition of two Orc expressions:

do f and g in parallel	$f \mid g$	Symmetric composition
for all x from f do g	$f > x > g$	Sequential composition
for some x from g do f	$f < x < g$	Pruning

Symmetric composition: $f \mid g$

- Evaluate f and g independently
- Publish all values from both
- No direct communication or interaction between f and g.
  They can communicate only through sites

Examples

- $CNN(d) \mid BBC(d)$: calls both CNN and BBC simultaneously
  Publishes values returned by both sites (0, 1 or 2 values)
- $WebServer() \mid MailServer() \mid LinuxServer()$
  May not publish any value

Sequential composition: $f \;\; >x> \; g$

For all values published by f do g
Publish only the values from g

- $CNN(d) \;\; >x> \; Email(address, x)$
  - Call $CNN(d)$
  - Bind result (if any) to x
  - Call $Email(address, x)$
  - Publish the value, if any, returned by $Email$

- $(CNN(d) \;|\; BBC(d)) \;\; >x> \; Email(address, x)$
  - May call $Email$ twice
  - Publishes up to two values from $Email$

Schematic of Sequential composition

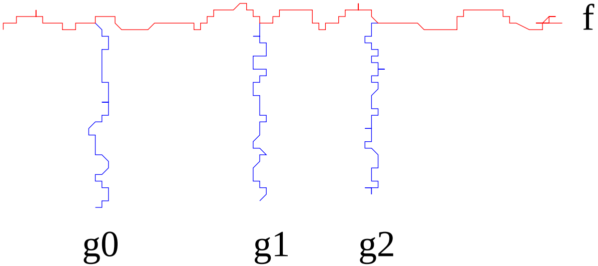


Figure: Schematic of $f \; >x> \; g$

Pruning: $(f \;{<}x{<}\; g)$

For some value published by g do f.

- Evaluate f and g in parallel.
  - Site calls that need x are suspended.
  - see $(M() \mid N(x)) \;{<}x{<}\; g$
- When g returns a (first) value:
  - Bind the value to x.
  - Terminate g.
  - Resume suspended calls.
- Values published by f are the values of $(f \;{<}x{<}\; g)$.

Example of Pruning

$$Email(address, x) \ {<}x{<} \ (CNN(d) \ | \ BBC(d))$$

Binds x to the first value from $CNN(d) \ | \ BBC(d)$.
Sends at most one email.

Some Fundamental Sites

- $if(b)$: boolean b,
  returns a signal if b is true; remains silent if b is false
- $Rtimer(t)$: integer t, $t \geq 0$, returns a signal t time units later
- $stop$: never responds. Same as $if(false)$
- $signal$: returns a signal immediately. Same as $if(true)$

Time-out

Publish M's response if it arrives before time t,
Otherwise, publish signal

$$z \ {<}z{<} \ (M() \ | \ Rtimer(t))$$

Some Target Applications

- Account management in a bank (Business process management):
  Workflow lasting over several months
  Security, Failure, Long-lived Data
- Extended 911:
  Using humans as components
  Components join and leave
  Real-time response
- Network simulation:
  Experiments with differing traffic and failure modes
  Animation
- Managing a city: (A proposal to EU)
  Components integrated dynamically
  The scope of software is nebulous

Expression Definition

$$\mathit{def}\ \ \mathit{MailOnce}(a) =$$
$$\mathit{Email}(a, m)\ \ <m<\ (\mathit{CNN}(d)\ \mid \mathit{BBC}(d))$$

$$\mathit{def}\ \ \mathit{MailLoop}(a, d) =$$
$$\mathit{MailOnce}(a)\ \gg\ \mathit{Rtimer}(d)\ \gg\ \mathit{MailLoop}(a, d)$$

- Expression is called like a procedure.
  It may publish many values. *MailLoop* does not publish
- Site calls are strict; expression calls non-strict

$$\mathit{def}\ \ \mathit{metronome}() = \ \mathit{signal}\ \mid (\mathit{Rtimer}(1)\ \gg \mathit{metronome}())$$
$$\mathit{metronome}()\ \gg\ \mathit{stockQuote}()$$

Orc as Programming Language

- Operators to Site calls:
  $1+(2+3)$ to $add(1,x) \ {<}x{<} \ add(2,3)$
- if E then F else G:
  $(if(b) \gg F \mid not(b) \ {>}c{>} \ if(c) \gg G) \ {<}b{<} \ E$
- $val \ x = G$ followed by F:
  $F \ {<}x{<} \ G$
- Data Structures, Patterns: Site calls and variable bindings
- Demo!

Laws Based on Kleene Algebra

(Zero and $\mid$)	$f \mid stop = f$
(Commutativity of $\mid$)	$f \mid g = g \mid f$
(Associativity of $\mid$)	$(f \mid g) \mid h = f \mid (g \mid h)$
(Idempotence of $\mid$) NO	$f \mid f = f$
(Associativity of $\gg$)	$(f \gg g) \gg h = f \gg (g \gg h)$
(Left zero of $\gg$)	$stop \gg f = stop$
(Right zero of $\gg$) NO	$f \gg stop = stop$
(Left unit of $\gg$)	$signal \gg f = f$
(Right unit of $\gg$)	$f > x > let(x) = f$
(Left Distributivity of $\gg$ over $\mid$) NO	$f \gg (g \mid h) = (f \gg g) \mid (f \gg h)$
(Right Distributivity of $\gg$ over $\mid$)	$(f \mid g) \gg h = (f \gg h \mid g \gg h)$

Adaptive Workflow

- This is no standard formal definition of workflow
  - Automated process with human participants
- Problem: How to upgrade active workflows?
  - Example: add time-out check/handler
- Van der Alst defined 20 Workflow patterns
  - Characterize workflow constructs
    - Sequence, Parallel, Split, Choice, Merge Instantiation, Termination, Milestone, ...
  - Can be expressed in Orc

The WfMC Basic Workflow Model

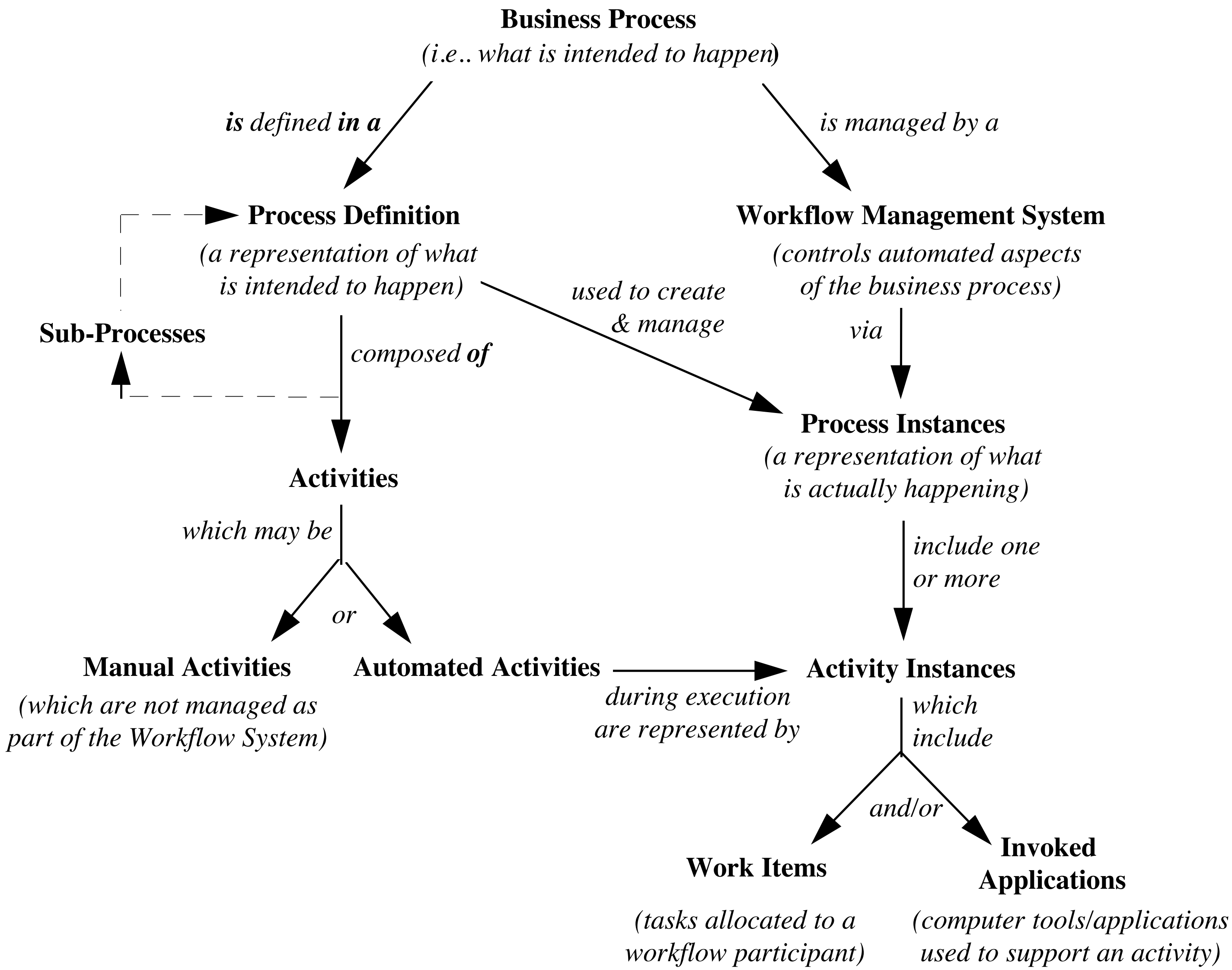


Figure 1 - Relationships between basic terminology

Figure from: Workflow Management Coalition. 1999. *Terminology & Glossary*. WfMC-TC-1011. p. 7.

Mapping WfMC Model → Orc

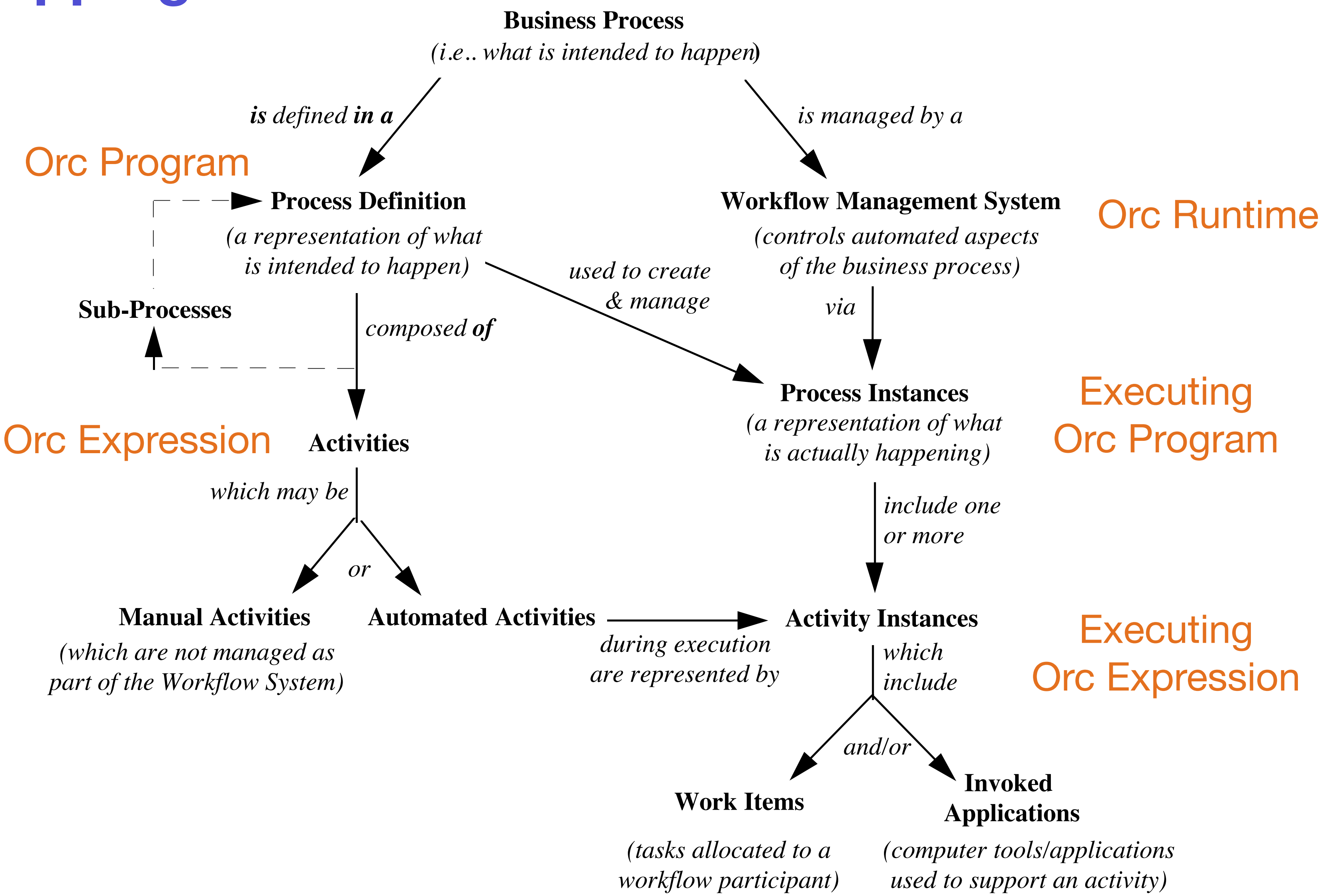


Figure 1 - Relationships between basic terminology

Types of Adaptive Workflow

- **Ad hoc**
  - ◻ Hand-upgrade active workflows
- **Systematic**
  - ◻ **Instantaneous**
    - Upgrade all active workflows at once
  - ◻ **Incremental**
    - Run old and new in parallel
    - Incrementally migrate processes
      - Wait for appropriate time

Wf Change Hazards

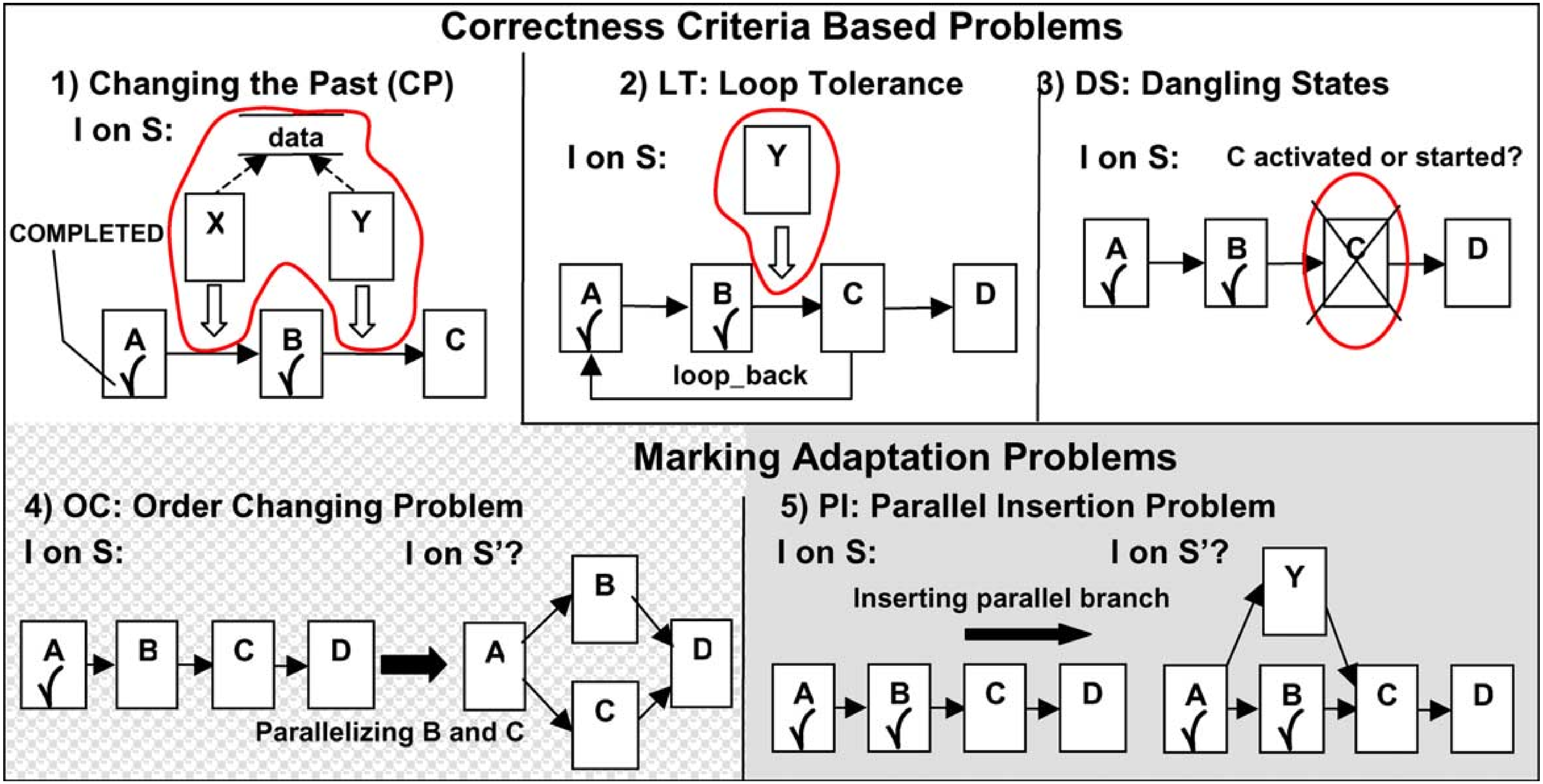
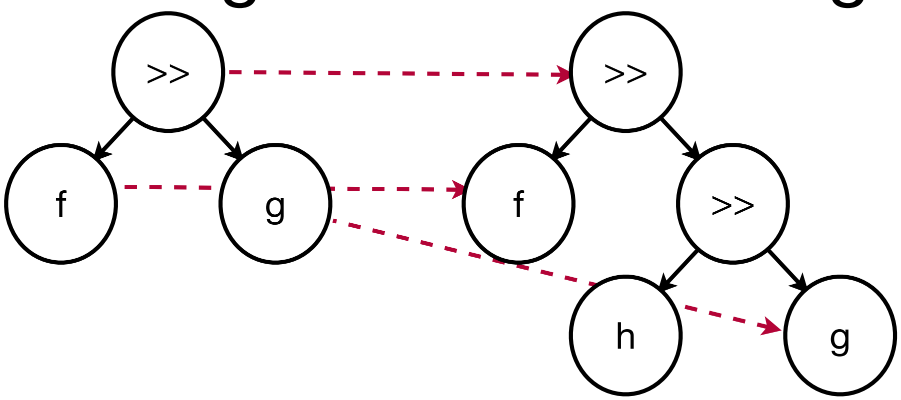


Fig. 3. Five typical problems regarding dynamic workflow change.

From RINDERLE, S., REICHERT, M., AND DADAM, P. 2004. Correctness criteria for dynamic changes in workflow systems: a survey. *Data Knowl. Eng. 50*, 1 (Jul. 2004), p. 15.

Adaptive Workflow in Orchard

- Define correspondence between old and new
  - f >> g ⇝ f >> h >> g



- Migrate active processes
  - Manage dependencies
  - Not necessarily behaviorally compatible

Secure Information Flow

How can confidential data be leaked?

Direct channels:

- *Site calls*
  - Memory
  - File system
  - Inter-process communication
  - Network access

Covert channels:

- *Control flow*
- *Exceptions*
- *Termination*
- *Timing*
- Scheduler
- Cache behavior
- Resource exhaustion
- Power

Secure Information Flow

Denning and Denning [1977].

- **Security labels**: associated with input and output variables. Form a lattice.
- **Non-interference**: High-security inputs cannot affect low-security outputs.
- **Information flow**: the value of one variable affects another.

Non-interference

Program P denotes a function on states S:

- $P : S \rightarrow S \cup \{\perp\}$

Equivalence relation $=_L$, where L is a label:

- $s =_L s'$ iff $\forall v \in L' \sqsubseteq L.\ s(v) = s'(v)$

Non-interference: for all input states s and s'

- $s =_L s' \Rightarrow P(s) =_L P(s')$

Kinds of Equivalence

- *Set* of possible outputs (possibilistic)
- *Distribution* of possible outputs (probabilistic)
- Program trace (observational determinism or bisimulation)

Language-Based Information Flow

Volpano, Smith and Irvine [1996] describe a type system for proving non-interference.

- Types are security labels
- Expression type: level of information revealed by its value
- Statement type: level of information revealed by the execution of the statement
- Well-typed terminating programs obey non-interference

Type System

As used by JIF, roughly.

Expression

$$\frac{\Gamma \vdash e : l \quad \Gamma \vdash e' : l'}{\Gamma \vdash e \star e' : l \sqcup l'}$$

Conditional

$$\frac{\Gamma \vdash e : l \quad l' = \Gamma(\mathsf{pc}) \sqcup l \quad \Gamma, \mathsf{pc} : l' \vdash a \quad \Gamma, \mathsf{pc} : l' \vdash b}{\Gamma \vdash \texttt{if } e \texttt{ then } a \texttt{ else } b}$$

Assignment

$$\frac{\Gamma(v) = l \quad \Gamma \vdash e : l' \quad \Gamma(\mathsf{pc}) \sqcup l' \sqsubseteq l}{\Gamma \vdash v := e}$$

Issues in Orc

Why can't Orc use such a type system?

- No separation between functional expressions and effectful statements.
- Data races can leak information.
- (Non-)Termination can leak information.

Examples

Halting

```
if h then l := true else l := false
```

which is sugar for:

```
  if(h) >> public(true)
| if(~h) >> public(false)
```

insecure due to non-termination!

Examples

Synchronization

```
Semaphore(0) >s>
public(false) >>
( s.acquire() >> public(true)
| if(h) >> s.release()
)
```

insecure due to non-termination!

Examples

Non-termination

```
def loop(x) =
  if ~x then loop(x)
       else Rtimer(1)

( public(false)
| loop(h) >> public(true)
)
```

insecure due to non-termination and data race!

Examples

Internal Timing

```
( Rtimer(50) >> public(true)
| (if h then Rtimer(100)
        else signal) >>
  public(false)
)
```

insecure due to data race!

Examples

External Timing

```
public(true) >>
(if h then Rtimer(100)
      else signal) >>
public(false)
```

insecure due to data race!

Typing Sites

What information goes into a site call?

- The fact that it was called
- The time that it was called
- The value passed for each argument

What information leaves a site call?

- The fact that it published
- The time that it published
- The value returned

Typing Expressions

Generalize the *pc* label used in static typing approach.

- What can we infer at any program point?
- We can infer that certain expressions published.
- Represent publication conditions as predicates over program variables.

Example 1

```
(if h then h' := true
      else h' := false) >>
l := true
```

Desugared:

```
( if(h) >> private(true)
| if(~h) >> private(false)
) >>
public(true)
```

Example 1

```
( if(h) >> private(true)
| if(~h) >> private(false)
) >>
public(true)
```

Assume that private always publishes.

$F \equiv$ `if(h) >> private(true)` publishes iff h.

$G \equiv$ `if(not(h)) >> private(false)` publishes iff $\neg h$.

F | G publishes iff $h \vee \neg h \equiv$ true.

Therefore $public(true)$ is always called and is secure.

Example 2

```
(if h then if(l) >> private(true)
      else private(false)) >>
public(true)
```

Desugared:

```
( if(h) >> if(l) >> private(true)
| if(~h) >> private(false)
) >>
public(true)
```

Example 2

```
( if(h) >> if(l) >> private(true)
| if(~h) >> private(false)
) >>
public(true)
```

$F \equiv$ `if(h) >> if(l) >> private(true)` publishes iff $h \wedge l$.
$G \equiv$ `if(not(h)) >> private(true)` publishes iff $\neg h$.
F | G publishes iff $(h \wedge l) \vee \neg h \equiv l \vee \neg h$.
Therefore calling $public(true)$ depends on the value of h and is insecure.

Summary

- Orc: Structured Concurrent Programming
  Direct representation of common concurrency structures
  Sites act as channels for more complex scenarios
- Applications to Workflow
  Organizing human-centric processes
  Direct representation of common concurrency structures
  Sites act as channels for more complex scenarios
  - Adaptive Workflow
    Updating a process that is running
  - Secure Information in Workflow
    Connecting information flow theory to practical workflow problems
- All work in progress

See http://orc.csres.utexas.edu