Workflow Patterns in Orc

William Cook
Sourabh Patwardhan
Jayadev Misra

Department of Computer Sciences
University of Texas at Austin

Overview of Orc

- Orchestration language
  - –Invoke services
  - –Manage time-outs, priorities, and failures
- Structured concurrent programming
  - –Implicit "result" channel, also explicit channels
  - –Easy to create and terminate processes
- Simple calculus with formal semantics
  - –Labeled transition system & traces
  - –Needs to be extended for time, etc.
- Prototype implementation available

Pipe: f >x> g

CNN >n> Email(user, n)

- –call CNN, bind result (if any) to n
- –then call Email to send news to user

- CNN and Email are sites
  - –Perform basic computations
  - –Return at most one value, or never return

Parallel: f | g

(CNN | BBC) >n> Email(user, n)

- –CNN | BBC may produce 0, 1, or 2 values

- Pipe also acts as "for each"
  - –Email is sent for *each* value
  - –Instances of g are executed in parallel
- Use f ≫ g if no variable is needed
  - –if f returns 1 value, acts as f; g

Where: f **where** x :∈ g

Email(user, n)
where n :∈ (CNN | BBC)

- –Binds n to *first* value of CNN | BBC
- –then *terminates* CNN | BBC

Some Basic Sites

let(x, …, z)	Returns argument values as a tuple
if(b)	Returns a signal if b is true; it **does not respond if b is false**
or(a, b) a + b	Sites, for basic operations
Rtimer(t)	Relative Timer: returns a signal after exactly t time units
0	never returns

Definitions: $E(x_1,\ldots,x_n) \triangleq f$

- Definitions
  First(g) ≜ let(x) **where** x :∈ g
- Using a definition
  First(CNN | BBC) >n> Email(user, n)

Subtleties: f **where** x :∈ g

- f and g are executed in ***parallel***
- Site calls are ***strict***
  - –not called until arguments are defined
- Example
  (M | N(x)) **where** x :∈ P
  - –M and P called immediately
  - –N is called after P returns

A larger example: Priority

- **If M responds in 10 time units, take its response, otherwise take first response**

Delay(N) ≜
    (Rtimer(10) ≫ let(u)) **where** u :∈ N

Priority(M, N) ≜ First(M | Delay(N))

- **Concise notation for**
  - –communication, blocking, and termination

Another example: Parallel Or

- **Return true if either M or N return true**

POr(M, N) ≜ let(z)
    where z :∈ if(x) | if(y) | or(x, y)
    where x :∈ M
    where y :∈ N

Orc Summary

e, f, g ::= c	constant
x	variable
$x(e_1, \ldots, e_n)$	call
f >x> g	pipe (also f ≫ g)
f | g	parallel
f **where** x :∈ g	asymmetric
$x(x_1, \ldots, x_n)$ ≜ f	definition

- **See web site for semantics, implementation**

`www.cs.utexas.edu/~wcook/projects/orc`

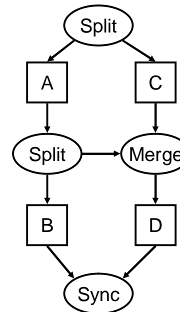
Workflow Patterns

- **Workflow products**
  - –use pictures to define workflows
  - –resemble Petri nets, Statecharts, concurrent flowcharts
- **No formal model of workflow**
- **Alternative: identify common *Patterns***
  - –20 patterns have been proposed
- **We show that Orc can implement the workflow patterns**

Simple Workflow Patterns

1. **Sequence**
   Seq(f, g) ≜ f ≫ g
2. **Parallel**
   Par(f*) ≜ f_1 | … | f_n
3. **Synchronization (fork-join):**
   Sync(f, g) ≜ let(x, y) **where** x :∈ f
                                where y :∈ g

Unstructured Workflows



Condition >M>
Sync(A ≫ M.set ≫ B,
         C ≫ M.wait ≫ D)

Condition creates a local object with two methods, set and wait, which blocks until set is called.

Choices

4. **Exclusive Choice (arbitration):**
   XOR(b, f, g) ≜ (if(b) ≫ f) | (if(¬b) ≫ g)
6. **Multi-choice:**
   Multi(b*, f*) ≜ Par(XOR(b_i, f_i, 0))
7. **Synchronizing Merge:**
   SyncMerge(b*, f*) ≜
        Sync(XOR(b_i, f_i, Signal))

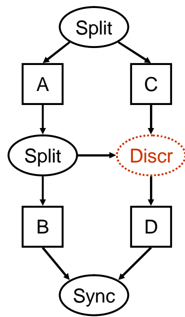
9. Discriminator

- **Produce result when first f_i completes**
  - –Continue other computations

Discr(f*) ≜ Buffer >S>
        S.get | (Par(f*) >x> S.put(x))

–Buffer creates a local object with two methods, put and get, which blocks until put is called.

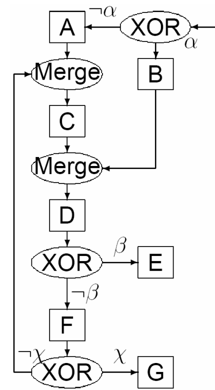
Modular Composition



Condition >M>
 Sync(A ≫ M.set ≫ B,
 Discr(C, M.wait) ≫ D)

D runs after C *or* A completes

10. Arbitrary Cycles



P ≜ XOR(α, PB, PA)
PA ≜ A ≫ PC
PB ≜ B ≫ PD
PC ≜ C ≫ PD
PD ≜ D ≫ XOR(β, E, PF)
PF ≜ F ≫ XOR(χ,G, PC)

12-15: Instantiation

- **Creating multiple instances of workflows**
  - 12. No synchronization
  - 13. Design-time knowledge
  - 14. Run-time knowledge
  - 15. No knowledge
- **These can be hard with Petri nets, but easy with a process calculus**

16 Deferred Choice

- **Let the environment make a choice by signaling an event**

DefChoiceTerm(e*, f*) ≜
 Which(e*) >k> Select(k, f*)

Which(e*) ≜
 First(e_1 ≫ let(1) | ⋯ | e_n ≫ let(n))

Select(k, f*) ≜
 if(k = 1) ≫ f_1 | ⋯ | if(k = n) ≫ f_n

17: Interleaved Parallel Routing

- Order is decided at run-time, and no two activities are executed at the same moment

Interleave(f*) ≜ Lock >M>
 (wait(M, f_1) | ⋯ | wait (M, f_n))

wait(M, f) ≜
 M.acquire ≫ f >x> M.release ≫ let(x)

–Lock creates a local object with two methods, acquire and release. Only one process may acquire the lock at a time

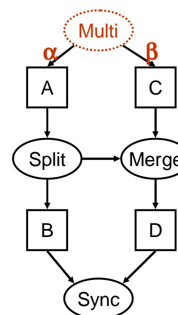
19/20: Cancel Activity/Case

- **Interrupt f when event e occurs**
  Interrupt(f, e) ≜ First(f | e)
- **Interrupt f after t time units, if it is not complete**
  Timeout(f, t) ≜ Interrupt(f, Rtimer(t))

Evaluation

- **Van der Aalst's concern with <u>encoding</u> is legitimate**
  - –Encodings are fine for theory
  - –But don't help programmers
  - –Require manual, global program rewrites
- **Orc uses definitions, not encodings**

Orc patterns are not fully compositional



Condition >M>
SyncMerge(
 ⟨α, A ≫ M.set ≫ B⟩,
 ⟨β, C ≫ M.wait ≫ D⟩)

–Does not work, because M.set is not called if α is false

–Problem: interaction between non-structured control and synchronizing merge

Related Work

- **van der Aalst**
  - –Defined initial set of 20 patterns
    - – not claimed to be complete
  - –Yet Another Workflow Language (YAWL)
    - – Add constructs to Petri nets until they can model the workflow patterns without "encoding"
- **Business Process Markup Language (BPML)**
  - –Pattern solutions similar to Orc
  - –More verbose, features missing
- **Workflow in π-Calculus (Puhlmann, Weske)**
  - –Uses explicit channels to signal start/end of workflows
  - –Incomplete model: termination requires encoding

Conclusions

- **Orc can implement all the workflow patterns**
- **Used definitions, avoids encoding**
  - –Still work to do: not completely compositional
- **Need to formalize more patterns**
  - –Parallel Or
  - –Priority
  - –Barrier syncrhonization