Structured Application Development over Wide-Area Networks

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

Internet Scripting

- 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.

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

- $+ \quad - \quad * \quad \&\& \quad || \quad < \quad =$ ...
- `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.

Overview of Orc

- Orc program has
  - a goal expression,
  - a set of definitions.
- The goal expression is executed. Its execution
  - calls sites,
  - publishes values.

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.

  Example: $CNN(d) \mid BBC(d)$

  calls both CNN and BBC simultaneously.
  Publishes values returned by both sites. (0, 1 or 2 values)

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 $Email$.

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

Notation: $f \gg g$ for $f \ >x> \ g$, if x unused in g.

Schematic of Sequential composition

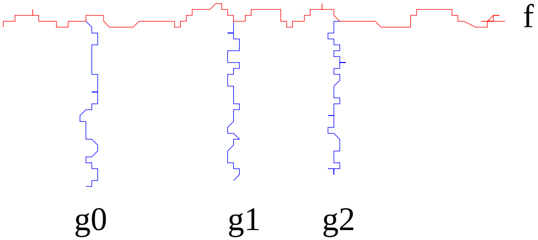

Figure: Schematic of f $>x>$ g

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 $if(true)$.

Centralized Execution Model

- An expression is evaluated on a single machine (client).
- Client communicates with sites by messages.

Expression Definition

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

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

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

$$\mathit{metronome}()\ \gg\ \mathit{stockQuote}()$$

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

Functional Core Language

- Data Types: Number, Boolean, String, with usual operators
- Conditional Expression: if E then F else G
- Data structures: Tuple and List
- Pattern Matching
- Function Definition; Closure

Variable Binding; Silent expression

$val\ \ x = \ 1 + 2$

$val\ \ y = \ x + x$

$val\ \ z = \ x/0$ -- expression is silent

$val\ \ u = \ \text{if } (0 < 5) \text{ then } 0 \text{ else } z$

Translating Functional Core to Pure Orc

- 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
- Function Definitions: Orc definitions

Comingling with Orc expressions

Components of Orc expression could be functional.
Components of functional expression could be Orc.

$$(1+2) \mid (2+3)$$

$$(1 \mid 2) + (2 \mid 3)$$

Convention: whenever expression F appears in a context where a single value is expected, convert it to $x \; {<}x{<} \; F$.

$1+2 \mid 2+3$ is $(x \; {<}x{<} \; add(1,2)) \mid (y \; {<}y{<} \; add(2,3))$

$(1 \mid 2) + (2 \mid 3)$ is $(add(x,y) \; {<}x{<} \; (1 \mid 2)) \; {<}y{<} \; (2 \mid 3)$

Example: Fibonacci numbers

$$
\begin{array}{l}
\mathit{def}\ \ H(0) = \ (1,1) \\
\mathit{def}\ \ H(n) = \ H(n-1)\ \ {>}(x,y){>}\ \ (y, x+y)
\end{array}
$$

$$\mathit{def}\ \ \mathit{Fib}(n) = \ H(n)\ \ {>}(x,\_){>}\ \ x$$

{- Goal expression -}

$\mathit{Fib}(5)$

Some Typical Applications

- Adaptive Workflow (Business process management):
  Workflow lasting over months or years
  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

Some Typical Applications, contd.

- Grid Computations
- Music Composition
- Traffic simulation
- Computation Animation

Some Typical Applications, contd.

- Map-Reduce using a server farm
- Thread management in an operating system
- Mashups (Internet Scripting).
- Concurrent Programming on Android.

Time-out

Publish M's response if it arrives before time t,
Otherwise, publish 0.

$$z \; <z< \; (M() \; | \; (Rtimer(t) \; \gg 0)), \text{ or}$$

$$\mathit{val} \;\; z = \; M() \; | \; (Rtimer(t) \; \gg 0)$$
$$z$$

Fork-join parallelism

Call M and N in parallel.
Return their values as a tuple after both respond.

$$
\begin{aligned}
&((u, v) \\
&\qquad < u < \; M()) \\
&\qquad < v < \; N()
\end{aligned}
$$

or,

$$(M(), N())$$

Recursive definition with time-out

Call a list of sites simultaneously.
Count the number of responses received within 10 time units.

$$\begin{array}{l} \mathit{def}\ \ \mathit{tally}([\,]) = 0 \\ \mathit{def}\ \ \mathit{tally}(M : MS) = (M() \gg 1 \mid \mathit{Rtimer}(10) \gg 0) + \mathit{tally}(MS) \end{array}$$

Barrier Synchronization in $M() \gg f \mid N() \gg g$

f and g start only after both M and N complete.
Rendezvous of CSP or CCS; M and N are complementary actions.

$$(M(), N()) \gg (f \mid g)$$

Priority

- Publish N's response asap, but no earlier than 1 unit from now. Apply fork-join between $Rtimer(1)$ and N.

$$val\ \ (u, \_) = (N(), Rtimer(1))$$

- Call M, N together. If M responds within one unit, publish its response. Else, publish the first response.

$$val\ \ x = M()\ \mid u$$

Interrupt f

Evaluation of f can not be directly interrupted.
Introduce a semaphore $interrupt$:

- $interrupt.release()$: to interrupt f
- $interrupt.acquire()$: responds after $interrupt.release()$ has been called.

Instead of evaluating

$$val\ \ z = f$$

evaluate

$$val\ \ (z, b) = f\ \ >x>\ (x, true)\ \ |\ \ interrupt.acquire()\ \ >x>\ (x, false)$$

Parallel or

Sites M and N return booleans. Compute their parallel or.

$$
\begin{array}{l}
val\ \ x = M() \\
val\ \ y = N() \\
\quad if(x) \gg true \mid if(y) \gg true \mid (x||y)
\end{array}
$$

To return just one value:

$$
\begin{array}{l}
val\ \ x = M() \\
val\ \ y = N() \\
val\ \ z = if(x) \gg true \mid if(y) \gg true \mid (x||y) \\
\quad z
\end{array}
$$

Airline quotes: Application of Parallel or

Contact airlines A and B.
Return any quote if it is below $300 as soon as it is available, otherwise return the minimum quote.
$threshold(x)$ returns x if $x < 300$; silent otherwise.
$Min(x, y)$ returns the minimum of x and y.

$$
\begin{array}{l}
val\ x = A() \\
val\ y = B() \\
val\ z = threshold(x)\ \mid\ threshold(y)\ \mid\ Min(x, y) \\
\quad z
\end{array}
$$

Backtracking: Eight queens

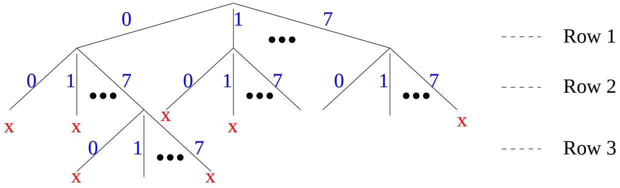

Figure: Backtrack Search for Eight queens

Eight queens; contd.

- xs: partial placement of queens (list of values from $0..7$)
- $extend(xs)$ publishes all solutions that are extensions of xs.
- $open(xs)$ publishes the columns that are open in the next row.
- Solve the original problem by calling $extend([])$.

$$
\begin{array}{l}
def\ \ extend(xs) = \\
\quad \text{if}\ \ (length(xs) = 8)\ \text{then}\ \ xs \\
\quad \text{else} \\
\qquad (open(xs)\ \ >j>\ extend(j : xs))
\end{array}
$$

Mutable Structures

$\text{val}\ \ r = Ref()$

$r.write(3)$, or $r := 3$
$r.read()$, or $r?$

$def\ \ swapRefs(x, y) = \ (x?, y?)\ >(xv, yv)>\ (x := yv, y := xv)$

Random Permutation

$val\ \ N = \ 20$ -- size of permutation array
$val\ \ ar = \ fillArray(Array(N), lambda(i) = i)$

-- Randomize array a of size n, $n \geq 1$
$def\ \ randomize(1) = \ signal$
$def\ \ randomize(n) =$
$\quad random(n)\ \ >k>$
$\quad swapRefs(ar(n-1), ar(k)) \gg randomize(n-1)$

$randomize(N)$

Binary Search Tree; Pointer Manipulation

$$
\begin{aligned}
&\mathit{def}\ \ \mathit{search}(\mathit{key}) = \ \ \text{-- return true or false}\\
&\quad \mathit{searchstart}(\mathit{key})\ >(\_,\_,q)>\ (q \neq \mathit{null})
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{def}\ \ \mathit{insert}(\mathit{key}) = \ \ \text{-- true if value was inserted, false if it was there}\\
&\quad \mathit{searchstart}(\mathit{key})\ >(p,d,q)>\\
&\quad \text{if}\ \ q = \mathit{null}\\
&\qquad \text{then}\ \ \mathit{Ref}()\ >r>\\
&\qquad\quad r := (\mathit{key},\mathit{null},\mathit{null})\ \gg \mathit{update}(p,d,r)\ \gg \mathit{true}\\
&\qquad \text{else}\ \ \mathit{false}
\end{aligned}
$$

$$
\mathit{def}\ \ \mathit{delete}(\mathit{key}) =
$$

Semaphore

$val\ \ s = \ Semaphore(2)$ -- s is a semaphore with initial value 2

$s.acquire()$
$s.release()$

Rendezvous:

$val\ \ s = \ Semaphore(0)$
$val\ \ t = \ Semaphore(0)$

$def\ \ send() = \quad t.release() \gg s.acquire()$
$def\ \ receive() = t.acquire() \gg s.release()$

n-party Rendezvous using $2(n-1)$ semaphores.

Readers-Writers

$$
\begin{array}{l}
val\ \ req = \ Buffer()\\
val\ \ cb = \ Counter()\\
\\
def\ \ rw() =\\
\quad req.get()\ \ >(b,s)>\\
\qquad (\ \ if(b) \quad \gg\ cb.inc() \qquad \gg s.release()\ \gg rw()\\
\qquad \ \mid if(\neg b) \ \gg\ cb.onZero()\ \gg\\
\qquad \quad cb.inc()\ \gg\ s.release() \quad \gg cb.onZero()\ \gg rw()\\
\qquad)\\
\\
def\ \ start(b) =\\
\quad val\ \ s = \ Semaphore(0)\\
\quad req.put((b,s))\ \gg s.acquire()\\
\\
def\ \ quit() = \ cb.dec()
\end{array}
$$

Processes

- Processes typically communicate via channels.
- For channel c, treat $c.put$ and $c.get$ as site calls.
- In our examples, $c.get$ is blocking and $c.put$ is non-blocking.
- Other kinds of channels can be programmed as sites.

Typical Iterative Process

Forever: Read x from channel c, compute with x, output result on e:

$$def\ \ P(c,e) = \ c.get() \ \ {>}x{>}\ \ Compute(x)\ \ {>}y{>}\ \ e.put(y)\ \ \gg\ P(c,e)$$

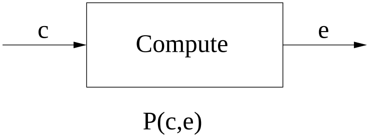

Figure: Iterative Process

Process Network

Process (network) to read from both c and d and write on e:

$$def\ \ Net(c,d,e) = \ P(c,e)\ \mid P(d,e)$$

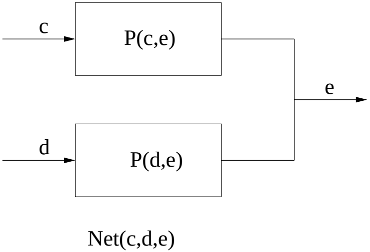

Figure: Network of Iterative Processes

Workload Balancing

Read from c, assign work randomly to one of the processes.

$$
\begin{aligned}
\textit{def} \;\; bal(c, c', d') = \;\; & c.get() \;\; >x> \; random(2) \;\; >t> \\
& (\text{if } t = 0 \text{ then } c'.put(x) \text{ else } d'.put(x)) \;\; \gg \\
& bal(c, c', d')
\end{aligned}
$$

$$
\begin{aligned}
\textit{def} \;\; WorkBal(c, e) = \;\; & val \;\; c' = \; Buffer() \\
& val \;\; d' = \; Buffer() \\
& bal(c, c', d') \;\; | \; Net(c', d', e)
\end{aligned}
$$

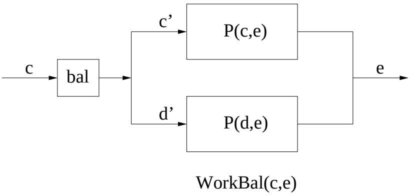

Figure: Workload Balancing in a network of Processes

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 $\mid$)	$(f \mid g) \gg h = (f \gg h \mid g \gg h)$

Additional Laws

(Distributivity over $\gg$) if g is x-free

$$((f \gg g) <x< h) = (f <x< h) \gg g$$

(Distributivity over $|$) if g is x-free

$$((f \mid g) <x< h) = (f <x< h) \mid g$$

(Distributivity over $<<$) if g is y-free

$$((f <x< g) <y< h)$$
$$= ((f <y< h) <x< g)$$

(Elimination of where) if f is x-free, for site M

$$(f <x< M) = f \mid (M \gg stop)$$