Structured Wide-Area Programming: Orc Calculus

Jayadev Misra

Department of Computer Science
University of Texas at Austin

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

- ubiquitous.
- difficult.
- important.
Some Typical Applications

- Map-Reduce using a server farm
- Thread management in an operating system
- Mashups (Internet Scripting)
- Reactive Programming
- Extended 911:
  Using humans as components
  Components join and leave
  Real-time response
Traditional approaches to handling Concurrency

- Adding concurrency to serial languages:
  - Threads with mutual exclusion using semaphore.
  - Transaction.
- Process Networks.
Features needed in a Concurrent Programming Language

- Describe entities and their interactions.
- Describe passage of time.
- Allow birth and death of entities.
- Allow programming of novel interactions.
- Support hierarchical structure.
Orc

- **Initial Goal**: Internet scripting language.
- **Next**: Component integration language.
- **Next**: A general purpose, structured “concurrent programming language”.
- **A very late realization**: A simulation language.
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.
Structured Concurrent Programming

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

- **Structured Concurrent Programming**: Component Integration in a concurrent world.
Philosophy of our Language Design

- Start with Concurrency. Add sequential features later.
- Impose hierarchical structure: compose concurrent programs.
- Introduce very few composition mechanisms (Combinators).
Orc Basics

- **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

New concepts are programmed using new sites.
Examples of Sites

- $+ - * \&\& |\| = \ldots$

- `Println`, `Random`, `Prompt`, `Email` ...  

- `Mutable Ref`, `Semaphore`, `Channel`, ...  

- `Timer`

- **External Services**: `Google Search`, `MySpace`, `CNN`, ...  

- `Any Java Class instance`, `Any Orc Program`  

- **Factory sites**: `Sites that create sites`: `Semaphore`, `Channel` ...  

- `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, $\text{CNN}(d)$
  Publishes the value returned by the site.

- **Composition** of two Orc expressions:

  \[
  \begin{align*}
  \text{do } f \text{ and } g \text{ in parallel} & \quad f | g \\
  \text{for all } x \text{ from } f \text{ do } g & \quad f > x > g \\
  \text{for some } x \text{ from } g \text{ do } f & \quad f < x < g \\
  \text{if } f \text{ halts without publishing do } g & \quad f ; g
  \end{align*}
  \]

  Symmetric composition

  Sequential composition

  Pruning

  Otherwise
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
    
    \[
    \text{Symmetric composition} \quad f | g
    \]
    
    for all \( x \) from \( f \) do \( g \)
    
    \[
    \text{Sequential composition} \quad f >x> g
    \]
    
    for some \( x \) from \( g \) do \( f \)
    
    \[
    \text{Pruning} \quad f <x< g
    \]
    
    if \( f \) halts without publishing do \( g \)
    
    \[
    \text{Otherwise} \quad f ; g
    \]
Structure of Orc Expression

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

- **Composition** of two Orc expressions:
  
  \[
  \begin{align*}
  \text{do } & f \text{ and } g \text{ in parallel} & f \mid g & \text{Symmetric composition} \\
  \text{for all } & x \text{ from } f \text{ do } g & f > x > g & \text{Sequential composition} \\
  \text{for some } & x \text{ from } g \text{ do } f & f < x < g & \text{Pruning} \\
  \text{if } & f \text{ halts without publishing do } g & f ; g & \text{Otherwise}
  \end{align*}
  \]
Structure of Orc Expression

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

- **Composition** of two Orc expressions:

  \[
  \text{do } f \text{ and } g \text{ in parallel} \quad f \| g \quad \text{Symmetric composition}
  \]
  \[
  \text{for all } x \text{ from } f \text{ do } g \quad f \gg x \gg g \quad \text{Sequential composition}
  \]
  \[
  \text{for some } x \text{ from } g \text{ do } f \quad f \ll x \ll g \quad \text{Pruning}
  \]
  \[
  \text{if } f \text{ halts without publishing do } g \quad f ; g \quad \text{Otherwise}
  \]
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  \[ f | g \]  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
  if *f* halts without publishing do *g*  \[ f ; g \]  Otherwise
Symmetric composition: $f \ | \ g$

- 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: $CNN(d) \ | \ BBC(d)$

Calls both $CNN$ and $BBC$ simultaneously. Publishes values returned by both sites. (0, 1 or 2 values)
Sequential composition: $f \gg x \gg g$

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

- $\text{CNN}(d) \gg x \gg \text{Email}(address, x)$
  - Call $\text{CNN}(d)$.
  - Bind result (if any) to $x$.
  - Call $\text{Email}(address, x)$.
  - Publish the value, if any, returned by $\text{Email}$.

- $(\text{CNN}(d) | \text{BBC}(d)) \gg x \gg \text{Email}(address, x)$
  - May call $\text{Email}$ twice.
  - Publishes up to two values from $\text{Email}$.

Notation: $f \gg g$ for $f \gg x \gg g$, if $x$ is 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.
    Consider $(M() \mid N(x)) < x < g$

- When $g$ returns a (first) value:
  - Bind the value to $x$.
  - Kill $g$.
  - Resume suspended calls.

- Values published by $f$ are the values of $(f < x < g)$. 
Example of Pruning

\[
Email(address, x) \leq x \leq (CNN(d) \mid BBC(d))
\]

Binds \( x \) to the first value from \( CNN(d) \mid BBC(d) \).
Sends at most one email.
Fork-join parallelism

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

$((u,v) < u < M()) < v < N())$
Otherwise: $f; g$

Do $f$. If $f$ halts without publishing then do $g$.

- An expression halts if
  - its execution can take no more steps, and
  - all called sites have either responded, or will never respond.

- A site call may respond with a value, indicate that it will never respond (helpful), or do neither.

- All library sites in Orc are helpful.
Examples of $f \; ; g$

1 ; 2 publishes 1

$\left( CNN(d) \mid BBC(d) \right) >x> Email(address, x) ; \text{Retry()}$

If the sites are not helpful, this is equivalent to

$\left( CNN(d) \mid BBC(d) \right) >x> Email(address, x)$
Some Fundamental Sites

- \textit{Ift}(b), \textit{Iff}(b): boolean \(b\),
  Returns a \textit{signal} if \(b\) is true/false; remains \textit{silent} otherwise.
  Site is helpful: indicates when it will never respond.

- \textit{Rwait}(t): integer \(t, \ t \geq 0\), returns a signal \(t\) time units later.

- \textit{stop}: never responds. Same as \textit{Ift}(false) or \textit{Iff}(true).

- \textit{signal}: returns a signal immediately.
  Same as \textit{Ift}(true) or \textit{Iff}(false).
Use of Fundamental Sites

- Print all publications of \( h \). When \( h \) halts, publish "done".
  \[ h \triangleright x \triangleright \text{Println}(x) \triangleright \text{stop} ; \text{"done"} \]

- Timeout:
  Call site \( M \).
  Publish its response if it arrives within 10 time units.
  Otherwise publish 0.

  \[ x \triangleleft x \triangleleft (M()) \mid \text{Rwait}(10) \triangleright 0 \]
Function Definition

```python
def MailOnce(a):
    Email(a, m) < m < (CNN(d) | BBC(d))

def MailLoop(a, t):
    MailOnce(a) >> Rwait(t) >> MailLoop(a, t)
```

- A function is called like a procedure. It may publish many values. *MailLoop* does not publish.
- Site calls are strict; Function calls non-strict.
Example of a Definition: Metronome

Publish a signal every unit.

\[
\text{def} \quad \text{Metronome}() = \text{signal} \quad | \quad (\text{Rwait}(1) \gg \text{Metronome}())
\]
Example of Function call

- Site $\text{Query}()$ returns a value (different ones at different times).

- Site $\text{Accept}(x)$ returns $x$ if $x$ is an acceptable value; it is silent otherwise.

- Call $\text{Query}$ every second forever and publish all its acceptable values.

\[
\text{Metronome}() \gg \text{Query}() >x> \text{Accept}(x)
\]
Concurrent function call

- Functions are often called concurrently.
- Each call starts a new instance of function execution.
- If a function accesses shared data, concurrent invocations may interfere.

**Example:** Publish each of "tick" and "tock" once per second, "tock" after an initial half-second delay.

```
Metronome() ⇒ ”tick”
| Rwait(500) ⇒ Metronome() ⇒ ”tock”
```
Laws about | and »

(Zero and | )  \( \text{f} | \text{stop} = \text{f} \)
(Commutativity of | )  \( \text{f} | \text{g} = \text{g} | \text{f} \)
(Associativity of | )  \( (\text{f} | \text{g}) | \text{h} = \text{f} | (\text{g} | \text{h}) \)
(Associativity of » ) if \( \text{h} \) is \( x \)-free
\( (\text{f} »x» \text{g}) »y» \text{h} = \text{f} »x» (\text{g} »y» \text{h}) \)
(Left zero of » )  \( \text{stop} » \text{f} = \text{stop} \)
(Left unit of » )  \( \text{signal} » \text{f} = \text{f} \)
(Right unit of » )  \( \text{f} »x» \text{x} = \text{f} \)
(Right Distributivity of » over | )
\( (\text{f} | \text{g}) »x» \text{h} = (\text{f} »x» \text{h} | \text{g} »x» \text{h}) \)

Identities that don’t hold

(Idempotence of | )  \( \text{f} | \text{f} = \text{f} \)
(Right zero of » )  \( \text{f} » \text{stop} = \text{stop} \)
(Left Distributivity of » over | )
\( \text{f} » (\text{g} | \text{h}) = (\text{f} » \text{g}) | (\text{f} » \text{h}) \)
Laws about \( \ll \)

(Right unit of \( \ll \)) \( f \ll stop = f \)

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

\[
(f \mid g) <x<h) = (f <x<h) \mid g
\]

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

\[
(f \gg g) <x<h) = (f <x<h) \gg g
\]

(Distributivity over \( \ll \)) if \( g \) is \( y \)-free and \( h \) is \( x \)-free

\[
(f \ll g) <y<h) \ll (f <y<h) \ll g
\]

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

\[
(f <x<M()> = f \mid (M()) \gg stop )
\]
Laws about ;

(Left unit of ; ) \( \text{stop} ; f = f \)

(Right unit of ; ) \( f ; \text{stop} = f \)

(Associativity of ; ) \( (f ; g) ; h = f ; (g ; h) \)