Structured Orchestration of Data and Computation

William Cook
Jayadev Misra
David Kitchin
John Thywissen

Department of Computer Science
University of Texas at Austin

http://orc.csres.utexas.edu

05/08/2012
HCSS, Annapolis

A Big Vision:
Software Challenge in the next two decades

- Design Methodology
    - Build it cheap
    - Build it reliable: Correctness, Fault-tolerance
    - Build it for evolution

- Security

Orc

- Orc addresses Design: as a component integration system.

  Components:

  - from many vendors
  - for many platforms
  - written in many languages
  - may run concurrently and in real-time

- Preliminary work on Security.

Evolution of Orc

- Web-service Integration
- Component Integration
- Structured Concurrent Programming

Web-service Integration: 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.

Component Integration

- Combine any kind of component, not just web services
- Small components: add two numbers, print a file ...
- Large components: Linux, MSword, email server, file server ...
- Time-based components: for real-time computation
- Actuators, sensors, humans as components
- Fast and Slow components
- Short-lived and Long-lived components
- Written in any language for any platform

Concurrency

- Component integration: typically sequential using objects
- Concurrency is ubiquitous
- Magnitude higher in complexity than sequential programming
- No generally accepted method to tame complexity
- May affect security

Traditional approaches to handling Concurrency

- Adding concurrency to serial languages:
  - Threads with mutual exclusion using semaphore.
  - Transaction.
- Process Networks.

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 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

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

  ...

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
if f halts without publishing do g	$f \; ; \; g$	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	$f \mid g$	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
if f halts without publishing do g	$f \; ; \; g$	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	$f \mid g$	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
if f halts without publishing do g	$f \; ; \; g$	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	$f \mid g$	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
if f halts without publishing do g	$f\ ;\ g$	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	$f \mid g$	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
if f halts without publishing do g	$f\ ;\ g$	Otherwise

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 is unused in g.

Schematic of Sequential composition

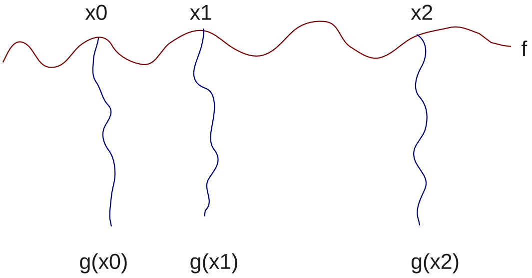


Figure: Schematic of $f \;>x>\; g$

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

$$Email(address, x) \;\; <x< \; (CNN(d) \mid BBC(d))$$

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

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.

Orc program

- Orc program has
    - a goal expression,
    - a set of definitions.

- The goal expression is executed. Its execution
    - calls sites,
    - publishes values.

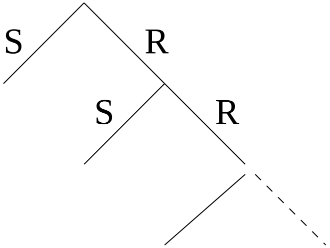
Some Fundamental Sites

- $\mathit{Ift}(b)$, $\mathit{Iff}(b)$: boolean b,
  Returns a signal if b is true/false; remains silent otherwise.
  Site is helpful: indicates when it will never respond.
- $\mathit{Rwait}(t)$: integer t, $t \geq 0$, returns a signal t time units later.
- *stop* : never responds. Same as $\mathit{Ift}(\mathit{false})$ or $\mathit{Iff}(\mathit{true})$.
- *signal* : returns a signal immediately.
  Same as $\mathit{Ift}(\mathit{true})$ or $\mathit{Iff}(\mathit{false})$.

Example of a Definition: Metronome

Publish a signal every unit.

$$\mathit{def}\ \ \mathit{Metronome}() = \underbrace{\mathit{signal}}_{S}\ \mid\ (\underbrace{\mathit{Rwait}(1) \gg \mathit{Metronome}()}_{R})$$



Publish an unending string of Random digits

$$Metronome() \gg Random(10)$$

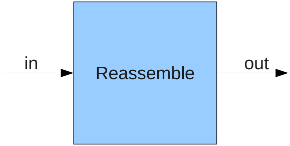
Power of Orc

- Solve all known synchronization, communication problems
- Code objects, active objects
- Solve all known forms of real-time and periodic computaions
- Solve a limited kind transactions
- and, all combinations of the above

Orc Language vs. Orc Calculus

- Data Types: Number, Boolean, String, with Java operators
- Conditional Expression: *if* E *then* F *else* G
- Data structures: Tuple, List, Record
- Pattern Matching; Clausal Definition
- Function Closure
- Comingling functional and Orc expressions
- Class for active objects

Packet Reassembly Using Sequence Numbers



in

Reassemble

out

$$\mathit{val}\ \ \mathit{ch} = \ \mathit{Table}(s, \mathit{lambda}(\_) = \mathit{Channel}())$$

$$\mathit{def}\ \ \mathit{read}() = \ \mathit{in.get}()\ \ >(n, v)>\ \mathit{ch}(n\%s).\mathit{put}(v)\ \gg \mathit{read}()$$

$$\mathit{def}\ \ \mathit{write}(i) = \ \mathit{ch}(i).\mathit{get}()\ \ >v>\ \mathit{out.put}(v)\ \gg \mathit{write}((i+1)\%s)$$

{- The ongoing computation -}

$\mathit{read}() \mid \mathit{write}(0)$

{- With Multiple Readers -} $\mathit{read}() \mid \mathit{read}() \mid \mathit{write}(0)$

Some Typical Applications

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

Some Very Large Applications

- Logistics
- Real time automotive software
- Large-scale hierarchical simulation
- Managing Olympic Games
- Smart City

Typical Computing Domains

- Software Integration within an organization
- Workflow
- Mediated Computing
- Perpetual Computing
- Rapid Prototyping

Current Status

- Strong Theoretical Basis
- An elegant programming language
    - as good as functional on functional problems
    - can work with mutable store, real-time dependent components
    - concurrency
    - hierarchical, modular, recursive
- Robust Implementation
    - Run program through a Web browser or locally
    - Web site: `orc.csres.utexas.edu`
    - Several papers, Ph.D. thesis
- Several Chapters of a book

Concurrent orchestration in Haskell

John Launchbury and Trevor Elliott
Proceedings of the third ACM Haskell symposium on Haskell

Large Scale Deployment

- Industrial strength Implementation
- Distributed Implementation
- Partnering