Living with Concurrency

Peter Grogono

Computer Science and Software Engineering
Concordia University

CUSEC 18 January 2008
Mathematics requirements in CS programs are shrinking.

The development of programming skills in several languages is giving way to cookbook approaches using large libraries and special-purpose packages.

The resulting set of skills is insufficient for today's software industry. . .

They quote. . .
Bjarne Stroustrup

“I have had a lot of complaints about [the use of Java as a first programming language] from industry, specifically from AT&T, IBM, Intel, Bloomberg, NI, Microsoft, Lockheed-Martin, and more.

 “[Texas A&M] did [teach Java as the first language]. Then I started teaching C++ to the electrical engineers and when the EE students started to out-program the CS students, the CS department switched to C++.”
At [XYZU], our [assembler and C++] students acquire and refine analytical and communications skills that make them better able to approach any problem creatively and successfully; the study habits and work ethic they develop are those needed for success in demanding graduate and professional programs and in real-world careers.
EDSAC II
Maurice Wilkes
“As soon as we started programming, we found to our surprise that it wasn’t as easy to get programs right as we had thought.

“Debugging had to be discovered.

“I can remember the exact instant when I realized that a large part of my life from then on was going to be spent in finding mistakes in my own programs.”
Java as the first language...

The good news is that students learn to
- program without distractions
- use libraries, packages, ...
- enjoy programming (pretty results)

The bad news is that students
- don't use algorithms
- are scared of pointers
- don't appreciate costs
“A cynic is a man who knows the price of everything but the value of nothing.”

Oscar Wilde (1854–1900)

“A LISP programmer knows the value of everything, but the cost of nothing.”

Alan Perlis (1922–1990)
default (Integer,Rational,Float)

infixr 9 #

series f = f : repeat 0

instance Num a => Num [a] where
    fromInteger c = series(fromInteger c)
    negate (f:ft) = -f : -ft
    (f:ft) + (g:gt) = f+g : ft+gt
    (f:ft) * gs@(g:gt) = f*g : ft*gs + series(f)*gt

instance Fractional a => Fractional [a] where
    (f:ft) / (g:gt) = qs where qs = f/g : series(1/g)*(ft-qs*gt)
    (f:ft) # gs@(0:gt) = f : gt*(ft#gs)
    revert (0:ft) = rs where rs = 0 : 1/(ft#rs)
    integral fs = 0 : zipWith (/) fs [1..]
    derivative (_:ft) = zipWith (*) ft [1..]

(Doug McIlroy, 1998)
\[
\frac{dy}{dx} = y \quad y(0) = 1
\]

\[
y = 1 + \int_0^x y(t) \, dt
\]

exp\(x\) = 1 + (\text{integral exp}x)

\[
\left[ \frac{1}{1}, \frac{1}{1}, \frac{1}{2}, \frac{1}{6}, \frac{1}{6}, \frac{1}{24}, \frac{1}{720}, \ldots \right]
\]
\[
\frac{d}{dx} \sin x = \cos x \\
\frac{d}{dx} \cos x = -\sin x
\]

\[
\sin x = \text{integral} \ \cos x
\]

\[
\cos x = 1 - (\text{integral} \ \sin x)
\]

\[
[1/1, -1/6, 1/120, -1/5040, \ldots]
\]

\[
[1/1, -1/2, 1/24, -1/720, \ldots]
\]
“I would like to see components become a dignified branch of software engineering.”

(Doug McIlroy, 1968)
Language is not the issue!

You should be able to think with:

- **objects**
  - C#, C++, Eiffel, Java, Smalltalk, . . .

- **functions**
  - LISP, Scheme, ML, Haskell, . . .

- **data**
  - lists, trees, graphs, maps, . . ., and corresponding algorithms

- **hardware**
  - registers, caches, addresses, pointers, . . .

- **concurrency**
  - processes, threads, semaphores, monitors, . . .
You have to know lots of other things too, of course, ...
Why Concurrency?
Why Concurrency?

[Dr. Dobbs Journal, 3/2005]
Why Concurrency?
Why Concurrency?

SPE = SIMD Synergistic Processor Element
Problem:

Concurrent programming is hard!

deadlock, livelock, starvation, race conditions, mamihlapinatapai, ...
<table>
<thead>
<tr>
<th>Presentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Logic</td>
</tr>
<tr>
<td>Data</td>
</tr>
</tbody>
</table>
The development of business applications using OO middleware has reached unparalleled complexity. In spite of greatly improved tools and development practices, more and more of the IT budget is wasted in maintenance rather than adding business value.

Dave Thomas (2008)
We must and can build concurrent computation models that are far more deterministic, and we must judiciously and carefully introduce nondeterminism where needed. Nondeterminism should be explicitly added to programs, and only where needed, as it is in sequential programming. Threads take the opposite approach. They make programs absurdly nondeterministic and rely on programming style to constrain that nondeterminism to achieve deterministic aims.

Edward Lee (2006)
Change moves upwards in the funnel:

- spaghetti $\rightarrow$ waterfall
- structured code $\rightarrow$ SADT
  - Structured Analysis & Design Technique
- object-oriented languages $\rightarrow$ OOAD
  - Object-Oriented Analysis & Design
- aspect-oriented languages $\rightarrow$ AOSD
  - Aspect-Oriented Software Development
Therefore:

To effect change in the software development process, we must change the programming paradigm.
Change moves upwards in the funnel:

- **spaghetti** → **waterfall**
- **structured code** → **SADT**
  Structured Analysis & Design Technique
- **object-oriented languages** → **OOAD**
  Object-Oriented Analysis & Design
- **aspect-oriented languages** → **AOSD**
  Aspect-Oriented Software Development
- **process-oriented languages** → **POMDD**
  Process-Oriented Model-Driven Design
Hypothesis

POMDD will succeed because:

- real world $\cong$ concurrent processes
- concurrent processes $\Rightarrow$ multiprocessors
- multiprocessors $\Rightarrow$ concurrent software
- concurrent software $\Rightarrow$ models real world

- cells/processes $\Rightarrow$ lower coupling
- lower coupling $\Rightarrow$ refactoring
- experience (1970s) $\Rightarrow$ we can do it!
The Erasmus Project

Desiderius Erasmus of Rotterdam (1466-1536)
The Erasmus Project
Main = ();

Main();
prot = [ ];
serverProc = { p +: prot | };
clientCell = ( p -: prot | );
Main = ( p :: prot; serverProc(p); clientCell(p) );
Main();
prot = [ start; *( query: Text; ^reply: Integer ); stop ];

serverProc = { p +: prot |
    p.start;
    loopselect
    || input: Text := p.query; p.reply := 0
    || p.stop; exit
    end
};

clientCell = ( p -: prot | );

Main = ( p :: prot; serverProc(p); clientCell(p) );

Main();
clientProc = \{ p -: prot | \};

clientCell = ( p -: prot | clientProc(p) );
Protocols

query = [ question; ^answer ]

sequence = [
  first: Integer;
  second: Text;
  third: Float ]

method1 = [ *( arg1; arg2; ...; ^result ) ]

method2 = [ *( arg1; arg2; ...; ^res1; ^res2 ) ]

class = [ *( M1 | M2 | ... | Mn ) ]
Statements

select
  || p.red; ...
  || p.yellow; ...
  || p.green; ...
end

select
  |stored < 10| buff[i] := p.x; ...
  |stored > 0| q.y := buff[j]; ...
end

select fair ...
select ordered ...
select random ...
Processes

prot = [ *( arg: Integer ) ];

filter = { p +: prot |
prime: Integer := p.arg;
sys.out := text prime + ' ';
q -: prot;
filter(q);
loop
n: Integer := p.arg;
if n % prime != 0
then q.arg := n
end
end
};
filter
Semantics vs. Deployment
Code

sqProt = [ *( query: Float; ^reply: Text ) ];
square = { p +: sqProt |
    loop
        q: Float := p.query;
        p.reply := text(q * q);
    end
};
squareCell = ( port +: sqProt | square(port) );
client = { p -: sqProt |
    p.query := 2;
    sys.out := p.reply + "\n";
};
clientCell = ( port -: sqProt | client(port) );
main = ( ch :: sqProt; squareCell(ch); clientCell(ch) );
main();
<Mapping>
  <Processor> alpha.encs.concordia.ca
    <Port> 5555 </Port>
    <Cell> squareCell </Cell>
    <Cell> clientCell1 </Cell>
  </Processor>
  <Processor> beta.encs.concordia.ca
    <Port> 5555 </Port>
    <Cell> squareCell1 </Cell>
    <Cell> clientCell </Cell>
  </Processor>
</Mapping>
Cells

- Programs consist of cells
- Cells may contain variables, processes, and cells
- Cells can be of any size
- Cells are “first-class citizens”
- Programs are “fractal”
- Control flow never crosses a cell boundary
- Cells are explicitly provided with all needed resources
- Cells may exchange messages
- Processes within a cell behave as co-routines
Processes

- A process is always inside a cell
- Processes may contain variables, processes, and cells
- Processes are "first-class citizens"
- All actions are performed within processes
- Control flow never crosses a process boundary
- A process may access variables within its cell
- Processes communicate by exchanging messages
- A process relinquishes control when it communicates

no race conditions
One program counter per cell
Protocols

- Protocols define interfaces
- Protocols specify communication patterns
- Protocols consist of typed messages and signals
- Protocols define sequence, choice, and repetition
- There is a “satisfaction” relation on protocols
Messages

- A “sent” message is an lvalue:
  
  ```
  p.result := 42;
  ```

- A “received” message is an rvalue:
  
  ```
  sum := p.val + ...;
  ```

- Signals synchronize:
  
  ```
  p.stop
  ```
Messages

- A "sent" message is an lvalue:
  
  ```
  p.result := 42;
  ```

- A "received" message is an rvalue:
  
  ```
  sum := p.val + q.val;
  ```

- Signals synchronize:
  
  ```
  p.stop
  ```
Separation of Concern

Cells define structure  ∼  Processes define action

Code defines meanings  ∼  Metacode defines deployment

Protocols specify processes  ∼  Protocols ensure satisfaction
The case against
The case against

It’s been tried before

(CSP, occam, Joyce, Amber, Erlang, Hermes, ...)
The case against

- It's been tried before
  
  (CSP, occam, Joyce, Amber, Erlang, Hermes, ...)

- Programming languages are not the problem
The case against

- It’s been tried before
  (CSP, occam, Joyce, Amber, Erlang, Hermes, ...)

- Programming languages are not the problem

- Object-oriented programming is good enough
The case against

- It’s been tried before
  (CSP, occam, Joyce, Amber, Erlang, Hermes, ...)
- Programming languages are not the problem
- Aspect-oriented programming is good enough
The case against

- It’s been tried before
  (CSP, occam, Joyce, Amber, Erlang, Hermes, ...)

- Programming languages are not the problem

- Aspect-oriented programming is good enough

- We’ve hidden the hard bits
  3-tier: CICS, J2EE, CORBA, ...
The case against

- It’s been tried before
  (CSP, occam, Joyce, Amber, Erlang, Hermes, ...)
- Programming languages are not the problem
- Aspect-oriented programming is good enough
- We’ve hidden the hard bits
  3-tier: CICS, J2EE, CORBA, ...
- Introducing a new paradigm is no longer feasible
The case for
The case for

- Many distributed applications
The case for

- Many distributed applications
- Multicore processors
The case for

- Many distributed applications
- Multicore processors
- Process-oriented programming is ... good
The case for

- Many distributed applications
- Multicore processors
- Process-oriented programming is ... good

We need software

development
maintenance
growth
adaptation
evolution
refactoring
The End

</Keynote>