Towards Malleable Software

Brian Shearing

Advanced Programming Specialist Group
British Computer Society, 14th April 2005

School of Computer Science & Information Systems
Birkbeck College, University of London, 3rd February 2005
### 1940s Machine Codes

**Kilburn Highest Factor Routine (amended)**

| Instruction | C 25 | 26 | 27 | Line | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|-------------|------|----|----|------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|
| -24 5 C    | -G1 | -  | -  | 1    | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| + 26      |     |    |    | 2    | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| -26 5 C    |     |    |    | 3    | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| + 27      |     |    |    | 4    | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| -23 5 C    | a   | Tn | -Gn| 5    | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Sub 27    |     |    |    | 6    | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Fct       |     |    |    | 7    |   |   |   | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Add 20 5 C|     |    |    | 8    | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Sub 26    |     |    |    | 9    | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Fct       |     |    |    | 10   | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Sub 25    |     |    |    | 11   | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Fct       |     |    |    | 12   | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Stop      |     |    |    | 13   |   |   |   | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| -26 5 C   |     |    |    | 14   | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Sub 21    |     |    |    | 15   | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Fct       |     |    |    | 16   | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Sub 27    |     |    |    | 17   | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| -27 5 C   |     |    |    | 18   | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Sub 26    |     |    |    | 19   | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Fct       |     |    |    | 20   |   |   |   | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Add 20 5 C|     |    |    | 21   | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Sub 25    |     |    |    | 22   | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Notes:**
- $C$ refers to the carry flag.
- $G_n$ refers to the $n$th carry.
- $T_n$ refers to the $n$th term in the sequence.
- $L_n$ refers to the $n$th load.

Tom Kilburn
Towards Malleable Software

1940s  Machine Codes
1950s  Assembly Languages
1960s  High-level Languages (Fortran, COBOL)
1970s  Structured Programming (Pascal, C)
1980s  Modular Programming (ADA, Modula-2)
1990s  Object-oriented Programming (C++, Java)
Towards Malleable Software

Hand wringing
Today’s software is not malleable

Finger wagging
What programming should be like

Chin stroking
Whence came the inflexibility, and where might malleability be found?

Arm waving
A model for malleable software

Head scratching
Research: verify or falsify the model
Part I

Today’s software is not malleable.
– What does this mean?
– Why does it matter?
Towards Malleable Software

Some transformations that require much programming

Fat client to or from thin one
Towards Malleable Software

Some transformations that require much programming

Fat client to or from thin one

Two-tier to three-tier

Single thread to multiplex

Cache sharing multiplex

© Brian Shearing & Peter Grogono  Towards Malleable Software, April 2005
Towards Malleable Software

Part II

What programming should be like
– with malleable software
– and tools adapted for malleability
Towards Malleable Software

Instructions to a compiler of a malleable notation:

- Compile Package *Product.Cache* in its own address space
- Compile *Policies.OvernightBatch* as a remote server using CORBA for communication
- Compile Process *Customer.Trolley* as a middleware component with multiple instances and automatic load-balancing
- Compile Components *Xml.Lexer* and *Xml.Parser* into one address space but as separate processes connected through a buffer with a capacity of 10,000 characters
Towards Malleable Software

A compiler that knows enough to obey the previous instructions also knows enough to obey these:

Show me the system's structure

Show the dataflows that result from me doing this

When I do that, where is the time spent?

in words, or in pictures

without having to buy separate tools from specialist vendors

It’s easy to do these things for malleable software
Towards Malleable Software

A compiler that knows enough to obey the previous instructions also knows enough to obey these:

- Build a reference manual for the system
- Build a book that explains how the system is put together
- Compile and run the system’s tests

JavaDoc, Eiffel and Dee show the way

‘Literate Programming’ and 3R show the way

JUnit shows the way, but test scripts must be in the source

About time too!
Towards Malleable Software

Part III

Whence came the inflexibility?
– What is it about our way of working that impedes malleability?
– Where might malleability be found?
A look at existing approaches that contribute to malleability
Object-oriented programming makes it possible to build systems that are readily adaptable and expandable — *within a given structure*.

It provides little help for altering structures themselves.

What we need is *refactoring in-the-large*. 
It’s a mess, and it doesn’t work.

But no block structure and no nested methods.
Towards Malleable Software

Every aspect’s interface must be public, even though each aspect should have its own access path.

Modular it isn’t! And for restructuring it’s hopeless.
Towards Malleable Software

**Principle of managed modularity**

An access permission is not a property of a component but of a relationship between components

*also known as*

The who’s-asking principle

*or as*

The principle of not calling out-the-back
‘The language designer should be familiar with many alternative features designed by others […]
One thing he should not do is to include untried ideas of his own. His task is consolidation, not innovation.’

C. A. R. Hoare

*Hints on programming language design in Computer Systems Reliability*
Towards Malleable Software

capabilities

Cambridge CAP computer

Maurice Wilkes

Roger Needham
HelloWorld:
    using (Standard) process(init: StandardIn)
declare
    Parms: StandardInterface;
begin
    receive Parms from Init;
    call Parms.PutLine(“Hello, World!”);
    return Parms;
end process

PutLine is a variable
Towards Malleable Software

Object style

Object

Source File Manager

Lexical Analyser

Syntactic Analyser

Code Generator

Binary File Manager

Process

Object style
Towards Malleable Software

**Compilation:**
- as is
- monolithic
- distributed

**Buffered:**
- Source File Manager
- Lexical Analyser
- Syntactic Analyser
- Code Generator
- Binary File Manager

**Object Style:**
- Source File Manager
- Lexical Analyser
- Syntactic Analyser
- Code Generator
- Binary File Manager

**Process Style:**
- Source File Manager
- Lexical Analyser
- Syntactic Analyser
- Code Generator
- Binary File Manager
Towards Malleable Software

process Producer (channel c) is
  loop  // 0
    p;  c ! α;  // 1
    q;  c ! β;  // 2
    r;
  end loop
end Producer

class Producer is
  integer state := 0;
  integer read() is
    loopcase state
      | 0:  p;  state := 1;
      | 1:  q;  state := 2;
      | 2:  r;  state := 0
    end loopcase
  end read
end Producer

process Consumer (channel c) is
  ... c ? v;
  ...
end Consumer

object style

Michael A. Jackson

process style

process Consumer (Producer c) is
  ...
  v := c.read();
  ...
end Consumer

inversion
Towards Malleable Software

Principle of process generality

Process-oriented programming is more general than object-oriented programming
Towards Malleable Software

Per Brinch Hansen

‘It is astounding to me that Java's insecure parallelism is taken seriously by the programming community, a quarter of a century after the invention of monitors and Concurrent Pascal.

It has no merit.

[...]

Java ignores the last twenty-five years of research in parallel languages.’

Letter to the editor
ACM Sigplan Notices
April 1999

shared variables, weak monitor semantics
Towards Malleable Software

Joyce (almost)

message protocol

type Stream is [int(integer), eos];

process sort(Stream i, o) is
  integer x;
  case
  | i ? eos: o ! eos
  | i ? int(x): subsort(x, i, o)
  end case
end sort;
Towards Malleable Software

type Stream is [int(integer), eos];

procedure subsort(integer x, Stream i, o) is
    Stream s := new Stream;  sort(s, o);
    integer y;
    loopcase
        | i ? eos:
            o ! int(x);  s ! eos;  exit loopcase
        | i ? int(y):
            if x > y then s ! int(x);  x := y else s ! int(y) end if
    end loopcase
end subsort;

message protocol
recursive process invocation
Joyce (almost)

process sort(Stream i, o) is
    integer x;
    case
        | i ? eos:  o ! eos
        | i ? int(x):  subsort(x, i, o)
    end case
end sort;
Towards Malleable Software

Remember the dotted lines

Compositional approach solves only half the problem; it composes but it does not deconstruct

Who can determine the point of discontinuity?

Today’s large system is tomorrow’s small

Principle of fractal construction

The same notation should be employed at all levels of scale
Scene: A design session

Dramatis personae:
Simon, a service object
Chloe, a client of Simon

Chloe: Please tell me the value of property p.
Simon: No.
Chloe: (Taken aback)
But I really need to know the value of p.
Simon: Why?
Chloe: Because I need to splonge it.
Simon: Just ask me, and I’ll splonge it for you.

Principle of property-less objects
Properties are not part of the object model

object-oriented programming!
Deciding if passing by reference is possible, safe, and advantageous is best left to software.
Towards Malleable Software

Part IV

A model for malleable software
– putting it all together
Towards Malleable Software

cell (encapsulation only, not relationships and not identity)
Towards Malleable Software

Cell (encapsulation only, not relationships and not identity)

Port = capability

Data, methods and processes all private

Only ports can be imported

One program counter per cell; processes within a cell are coroutines

c.f. JSR 121 ‘Isolates’

Model for malleability
Towards Malleable Software

Part V

Research to verify or falsify the model.
– Develop and prove the flexibility, simplicity and efficiency of the model
Towards Malleable Software

- complete first draft of model
- demonstrate mapping of business models
- model => abstract syntax => concrete syntax
- simple compiler and interpreter for notation
- configurability
- process-oriented library, inc. GUI
- code generator
- IDE, with visualisation tools
Towards Malleable Software

- complete first draft of model
- demonstrate mapping of business models
  - model => abstract syntax => concrete syntax
- simple compiler and interpreter for notation
- configurability
- process-oriented library, inc. GUI
- code generator
- IDE, with visualisation tools

essential for performance
- e.g. communication: Java 10,000i, Hermes 9i

build complete system as is, monolithic, distributed ...
make random cuts; put extracts on remote machines
compiler (in malleable notation) compiles at 1,000 ipc
Towards Malleable Software

1940s  Machine Codes
1950s  Assembly Languages
1960s  High-level Languages
1970s  Structured Programming
1980s  Modular Programming
1990s  Object-oriented Programming
2000s  ???
## Towards Malleable Software

<table>
<thead>
<tr>
<th>Decade</th>
<th>Programming Style</th>
</tr>
</thead>
<tbody>
<tr>
<td>1940s</td>
<td>Machine Codes</td>
</tr>
<tr>
<td>1950s</td>
<td>Assembly Languages</td>
</tr>
<tr>
<td>1960s</td>
<td>High-level Languages</td>
</tr>
<tr>
<td>1970s</td>
<td>Structured Programming</td>
</tr>
<tr>
<td>1980s</td>
<td>Modular Programming</td>
</tr>
<tr>
<td>1990s</td>
<td>Object-oriented Programming</td>
</tr>
<tr>
<td>2000s</td>
<td>Process-oriented Programming</td>
</tr>
</tbody>
</table>
Towards Malleable Software

1940s Machine Codes
1950s Assembly Languages
1960s High-level Languages
1970s Structured Programming
1980s Modular Programming
1990s Object-oriented Programming
2000s Process-oriented Programming
2010s Malleability: Modular Concurrency