A High-Performance Operating System for Structured Concurrent Programs

Luc Bläser
ETH Zurich, Switzerland
blaeser@inf.ethz.ch
State of the Art

Main problems of object-oriented languages:

• References
  – Arbitrary object interlinking => Unstructured dependencies
  – No hierarchical composition => Objects can not encapsulate (dynamic) structures of other objects

• Threads
  – Concurrency only added with hindsight to procedural programming model
  – Threads operate on arbitrary objects via method calls => error-prone
New Approaches

Trends towards improved programming models:

• First-class concurrency
  – Concurrency as primary language construct in the form of self-active objects / components
  – Message communication instead of blocking method calls
  – Examples: Active C#, Zonnon, Composita

• Pointer-free structuring
  – Hierarchical composition instead of flat object graph
  – Hierarchically controlled interface wiring
  – Examples: ArchJava, Classages, Composita

⇒ New requirements for modern runtime systems
Modern Runtime Systems

Requirements:

• Highly-scalable concurrency
  – Support of a very high number of light-weighted processes

• High-performance concurrency
  – Efficient execution of highly-interactive concurrent programs

• Efficient memory management
  – Efficient and safe memory management exploiting improved program structures

• Liberation from artifacts
  – Abandon system features that are no longer needed for a modern programming model
Example of Structured Concurrency

The Component Language

• Components as general abstraction units
• Strict encapsulation
  – External dependencies only via explicit interfaces allowed
• A component can offer and require interfaces
  – offered interfaces represent own facts of the component
  – required interfaces are to be offered by other components
• Multi-instantiation from component templates

COMPONENT Producer
REQUIRES DataAcceptor;
END Producer;

COMPONENT BoundedBuffer
OFFERS DataAcceptor,
DataSource;
END BoundedBuffer;

COMPONENT Consumer
REQUIRES DataSource;
END Consumer;

Producer
\----\ Data-Acceptor
  \----\ Data-Acceptor
    \----\ BoundedBuffer
      \----\ DataSource
        \----\ Consumer
Component Relations

- Hierarchical composition

- Interface connections

- Communication-based interactions

encapsulated sub-components

connection between required & offered interface

bidirectional communication

concurrent components
Hierarchical Structuring

COMPONENT Simulation
VARIABLE
  buffer: BoundedBuffer;
  producer[i: INTEGER]: Producer;
  consumer[k: INTEGER]: Consumer;
BEGIN
  NEW(buffer);
  FOR i := 1 TO user input N DO
    NEW(producer[i]); CONNECT(DataAcceptor(producer[i], buffer)
  END;
  FOR k := 1 TO user input M DO
    NEW(consumer[i]); CONNECT(DataSource(consumer[i], buffer)
  END
END Simulation;

containers for components
dynamic collection
dynamic construction

network structure exclusively controlled by surrounding component
COMPONENT Producer
    REQUIRE DataAcceptor;
BEGIN
    FOR i := 1 TO N DO
        DataAcceptor!Element(i)
    END;
    DataAcceptor!Finished
END Producer;

COMPONENT BoundedBuffer
    OFFERS DataAcceptor;
IMPLEMENTATION DataAcceptor;
BEGIN {EXCLUSIVE}
    WHILE ?Element DO
        AWAIT(empty);
        ?Element(x); empty := FALSE
    END;
    ?Finished
END DataAcceptor;
END BoundedBuffer;
Component Operating System

High-performance runtime system for structured concurrent programs of the component language

Highlights:

• Light-weighted processes
  – Micro stacks with size that can dynamically grow and shrink
  – Enables very high number of processes

• Fast context switches
  – Direct synchronous context switches
  – Low-cost and efficient preemption based on code-instrumentation

• Safe and efficient memory management
  – Garbage collection no longer needed due to hierarchical structures
  – Virtual memory management not needed
Stack Management

- Arbitrarily small stack sizes (not fixed to pages)
- Uniformly represented as heap blocks
- No method calls
  - Stacks only grow due to component-internal procedures
- System calls run on processor-associated system stacks
- Dynamic growing and shrinking
  - Compiler-inserted checks at procedure entry and exit
Process Management

- Direct context switches within monitors and communication

- Software-controlled preemption
  - Compiler automatically inserts checks in the machine code
  - Checks are executed in guaranteed small time intervals
  - Checks initiate preemption after a defined time interval
  - No cooperative multi-task programming

- Save only the necessary registers on preemption
  - Mostly no temporary registers used between statements
  - Economize extensive register backup space for each process
  - Runtime overhead of instrumented checks about 0.5%

```assembly
CMP EDI, 0
JL continue
save used registers
CALL Preempt
restore used registers
continue:
```
Communication Channel

- Bounded FIFO message buffer for each communication
- Automatic communication start and termination
  - Start by the first message command
  - Termination on the last protocol transition or on component finalization
- Dynamic protocol monitoring

```
INTERFACE Hotel
{
   IN CheckIn
   (OUT AssignedRoom
    IN CheckOut OUT Bill
    [ IN DirectPayment ]
    OUT FullyBooked
   )
}
END Hotel
```
Memory Management

• Hierarchy of component networks
  – Arbitrary $n$-to-$m$ component networks within each component possible
  – Interface connections solely set by the surrounding component

• Hierarchical lifetime dependencies
  – Deletion of a component => automatic deletion of the sub-components
  – Explicit deletion of a component => outer interfaces of the component are safely disconnected

=> Safe memory management without garbage collection
Scalability und Performance

- Maximum number of processes (threads)

<table>
<thead>
<tr>
<th>Component OS</th>
<th>C#</th>
<th>Java</th>
<th>Oberon AOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>5'010'000</td>
<td>1'890</td>
<td>10'000</td>
<td>15'700</td>
</tr>
</tbody>
</table>

4GB main memory, City Benchmark

- Execution performance

<table>
<thead>
<tr>
<th>Program (in sec)</th>
<th>Component OS</th>
<th>C#</th>
<th>Java</th>
<th>Oberon AOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>City</td>
<td>0.24</td>
<td>0.66</td>
<td>440</td>
<td>4.1</td>
</tr>
<tr>
<td>ProducerCons.</td>
<td>16</td>
<td>19</td>
<td>130</td>
<td>60</td>
</tr>
<tr>
<td>Eratosthenes</td>
<td>1.8</td>
<td>6.8</td>
<td>4.6</td>
<td>5.8</td>
</tr>
<tr>
<td>News</td>
<td>2.0</td>
<td>3.5</td>
<td>3.9</td>
<td>4.6</td>
</tr>
<tr>
<td>TokenRing</td>
<td>2.1</td>
<td>22</td>
<td>22</td>
<td>18</td>
</tr>
<tr>
<td>Mandelbrot</td>
<td>0.88</td>
<td>0.43</td>
<td><strong>0.39</strong></td>
<td>0.6</td>
</tr>
<tr>
<td>TrafficSimulation</td>
<td><strong>0.05</strong></td>
<td>33</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

6 CPUs Intel Xeon 700MHz, C# & Java Windows Server Enterprise Edition
C#, Java, AOS: analogous programs with methods instead of communication
Predictability without Garbage Collection

GC Peaks

Oberon AOS
Component OS

No GC needed

500 subsequent executions of the Small City program (in ms)
Conclusions

A new efficient runtime system for a modern structured concurrent programming language

• Technical benefits
  – High scalability in the number of processes
  – Fast execution of concurrent programs
  – No garbage collection needed
  – Customized and optimized for structured concurrency

• Conceptual benefits
  – Hierarchical controlled structured and guaranteed encapsulation
  – Inherent concurrency with communication-based interactions

• Practical application in traffic simulation (TU Berlin)

• Project Website: http://www.jg.inf.ethz.ch/components