Improving Aglets with Strong Agent Mobility through the IBM JikesRVM

Raffaele Quitadamo,
G. Cabri, L. Ferrari, L. Leonardi

DII - University of Modena and Reggio Emilia - ITALY

quitadamo.raffaele@unimore.it

http://www.agentgroup.unimore.it
Outline

- Introduction
- Strong mobility vs. weak mobility
- IBM JikesRVM
- The strong Aglets prototype
- Conclusions and future work
A few words on code mobility...

- A new paradigm to tackle complexity in the design of distributed applications
- The “mobile component” can dynamically relocate its bindings with its hosting execution environment and re-establish them elsewhere

A new breed of possibilities but also new challenging issues to deal with

**Weak mobility**
- lightweight migrable agents
- no execution state preservation
- uncomfortable programming model

**Strong mobility**
- agent interrupted and resumed transparently
- execution state saved (stack, PC, ..)
- very expensive and hard to implement

**Full mobility**
- migration of processes with their kernel-mediated state
- used in distributed O.S. (LOCUS...)
- not well-suited to Java agents
How to implement strong mobility?

Research in this field proposes two approaches (mainly for Java agents):

Modifying JVM to externalize the agent’s state

1. Introduce into the original VM some state capturing and restoring facilities
2. Intrusive modifications introduce three kinds of problems:
   - Trust
   - Security
   - Performance
3. Most VMs are written in non Java languages

Manipulating the agent’s code

1. Intrinsic portability
2. Injection of hidden control instructions to trace execution state
3. Limitations due to:
   - Incomplete state capturing
   - Fixed migration points
   - Performance
4. Useful for simulating strong mobility power with weak mobility (e.g. X-KLAIM)
IBM Jikes Research Virtual Machine

- A research project of IBM T. J. Watson

- **Objective**: “exploring the possibilities that a VM, written almost completely in Java, can offer”

- JikesRVM’s core is
  1. made up of all Java objects (threads, stacks, context registers, …)
  2. portable among different architectures (IA32, PPC32, PPC64) and OS (Linux, AIX and Mac OS)
  3. Available at [http://jikesrvm.sourceforge.net](http://jikesrvm.sourceforge.net) as a fully open-source project
Why choosing JikesRVM?

JikesRVM was chosen because:

1. Easily **extensible** with new Java classes without affecting the core runtime classes

2. Targeted to large multi-processor servers (SMP)

3. Designed to support thousands of threads with high **scalability** and **performance** levels

4. Threads = **lightweight objects** scheduled internally by the VM, **without** using native OS threads

5. Built-in facilities exploited to easily capture the execution state:
   - OSR (On Stack Replacement)
   - Type-accurate GC (Garbage Collector)
Thread’s state capturing

JikesRVM Thread

Take yield point & process dispatch message

Invoke the migration service and auto-suspend

MobileFrame for Method A
- Bytecode index
- Method signature (full name, descriptor)
- Array of local variables
- Array of stack operands

OSR is triggered and each stack frame is captured in an OSR scope descriptor

OSR scope descriptor converted in MobileFrame
(a serializable, bytecode-level representation of the stack frame)

The chain of MobileFrame is serialized with the Thread object into a Java socket
A new thread is instantiated locally and gets auto-suspended.

The frames are injected into the stack of the new thread:

1. Objects are de-serialized into the heap
2. Methods are compiled (if not yet compiled)

The thread is resumed locally transparently.
Aglets: project originally developed by the IBM Tokyo Research Laboratory

- Successful platform for the development of mobile agent based applications
- 100% Java library
- Weak mobility provided through Java serialization
- special method `dispatch(..)` to move to another host
- each time an agent is resumed, its execution restarts from a defined entry point (= `run()` method)
- Aglets is now an open-source project (http://aglets.sourceforge.net)
Main objectives:

1. creating a MAP with the simplicity and intuitiveness of Aglets, but the power of strong mobility

2. evaluating the tradeoff between benefits and overheads

Strong Aglets

- JikesRVM thread assigned to each aglet
- The thread is the only handler of the messages posted to the aglet

Current Aglets

- a “thread pool” to contain Java threads that will run into the methods of the aglet
- Each aglet can be served by many threads in the system
Our Aglets prototype (2)

Strong Aglets

- The aglet’s methods **can be interrupted** by the *dispatch* message
- **transparent migration**
- **execution state** is **preserved** by transmitting the sequence of frames
- The aglet is **resumed** remotely **from the last instruction** executed

Current Aglets

- The *dispatch* message **cannot interrupt** the current method
- **Execution state** is **lost**
- Only the **serializable fields** of the aglet are saved (intermediate results, locals, …)
- Restarts from the beginning of run()
Our Aglets prototype (3)

- **JikesRVM yield points** are used to make the thread lose control and test if a dispatch is requested

- **Yield points** are a built-in feature of the **quasi-preemptive** JikesRVM scheduler

- User can choose which methods can be interrupted by the *dispatch* notification

- The aglet can also invoke the migration by itself, using the *dispatch()* method

The Aglets programming model is left as much as possible unchanged!
Other ongoing features

- Data-space relocation strategies using “object policies”, that the user can register for specific objects (e.g. System.err, System.in, …)

- Migration of objects with associated locks

- Migration of multi-threaded agents, with mutual dependencies
Some preliminary performance tests

<table>
<thead>
<tr>
<th></th>
<th>5 frames</th>
<th>15 frames</th>
<th>25 frames</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frame extraction</td>
<td>1.78E-5</td>
<td>1.89E-5</td>
<td>1.96E-5</td>
</tr>
<tr>
<td>State building</td>
<td>3.44E-5</td>
<td>3.75E-5</td>
<td>3.43E-5</td>
</tr>
<tr>
<td>Pure serialization</td>
<td>2.49E-3</td>
<td>7.32E-3</td>
<td>1.50E-2</td>
</tr>
<tr>
<td>Overall times</td>
<td>2.54E-3</td>
<td>7.38E-3</td>
<td>1.51E-2</td>
</tr>
</tbody>
</table>

- The main part of the migration time is due to the pure Java serialization
- State capturing with the OSR is very fast

<table>
<thead>
<tr>
<th></th>
<th>5 frames</th>
<th>15 frames</th>
<th>25 frames</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pure deserialization</td>
<td>4.46E-3</td>
<td>5.33E-3</td>
<td>7.06E-3</td>
</tr>
<tr>
<td>State rebuilding</td>
<td>5.45E-4</td>
<td>5.27E-4</td>
<td>5.06E-4</td>
</tr>
<tr>
<td>Stack installation</td>
<td>1.53E-3</td>
<td>1.60E-3</td>
<td>1.71E-3</td>
</tr>
<tr>
<td>Overall times</td>
<td>6.54E-3</td>
<td>7.46E-3</td>
<td>9.28E-3</td>
</tr>
</tbody>
</table>
The main advantages of our approach

- **Written entirely in Java**, thus portable as much as possible

- **No overhead** is imposed on the execution of threads, because the **OSR extraction** is performed only **at migration time** and with very short times ($10^{-5}$ sec)

- The **JVM** is **not modified** at all, but simply **extended** with a Java package

- Other parts of JikesRVM are not influenced by our system, thus avoiding security problems and performance degradation
Limitations and open research issues

- The OSR extraction mechanism can be used to capture those frames that are baseline compiled. However, JikesRVM can be built with two additional JIT compilers (the quick compiler and the optimizing compiler).

- In the future, we will try to offer the possibility of capturing optimized frames too.

- The bottleneck of the serialization is being carefully considered to reduce minimize the migration and restoration times.

- Additional comparisons tests will be performed with the current Aglet release.

- Available at http://agentgroup.unimo.it as soon as possible.
Conclusions

- We have developed an “Aglets fork” that is enriched with strong mobility power

- Thread state capturing and restoring was achieved through the exploitation of JikesRVM advanced technologies (OSR, ...)

- We found in IBM JikesRVM a lot of built-in facilities that, as we are experimenting, may be of crucial importance to those who develop Mobile Agents Platforms with strong mobility

- The Aglets interface was left unchanged, resulting very familiar to Aglets programmers
Thanks for your attention!