Fully Time Deterministic Java™ Technology

Jean-Marie Dautelle, Senior Principal Engineer
Raytheon Company

TS-4797

Java

Sun microsystems

- Because real-time programming requires a time-predictable standard library.

GOAL

Biography

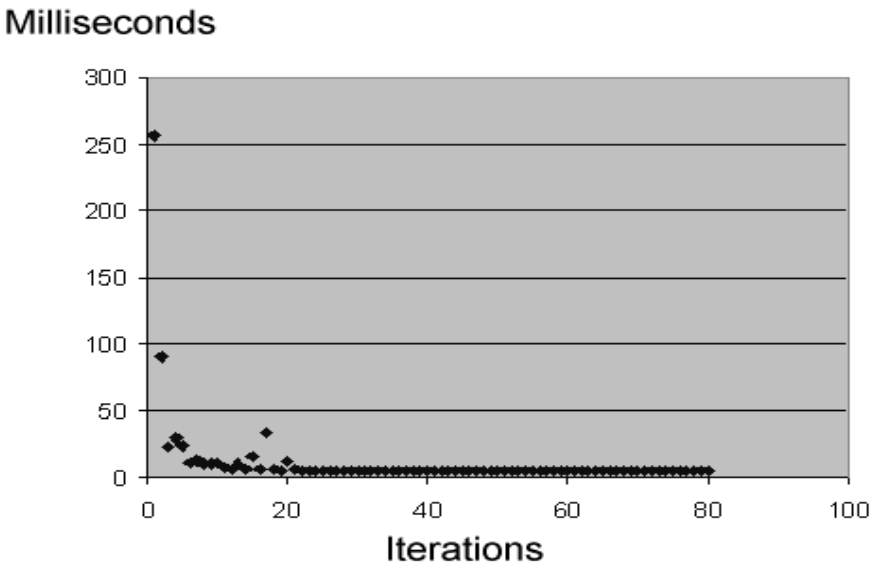
- Jean-Marie holds a master degree of electrical engineering (Orsay University, France) and a post graduate degree in Information Processing and Simulation.
- Jean-Marie is a Java Executive Committee member (as Individual) since November 2006 (Java™ 2 Platform, Micro Edition)
- Jean-Marie has published articles in major computing magazines (e.g. Java Developer Journal, Dr Dobbs, JavaWorld™ publication)
- Jean-Marie is the project leader and main author of the **J**avolution framework (http://javolution.org) and the **JS**cience library (http://jscience.org).

Agenda

- Current Challenges
- The Standard Library
- A Real-Time Library
- Context Programming
- Real-Time I/O
- Conclusion

Current Challenges

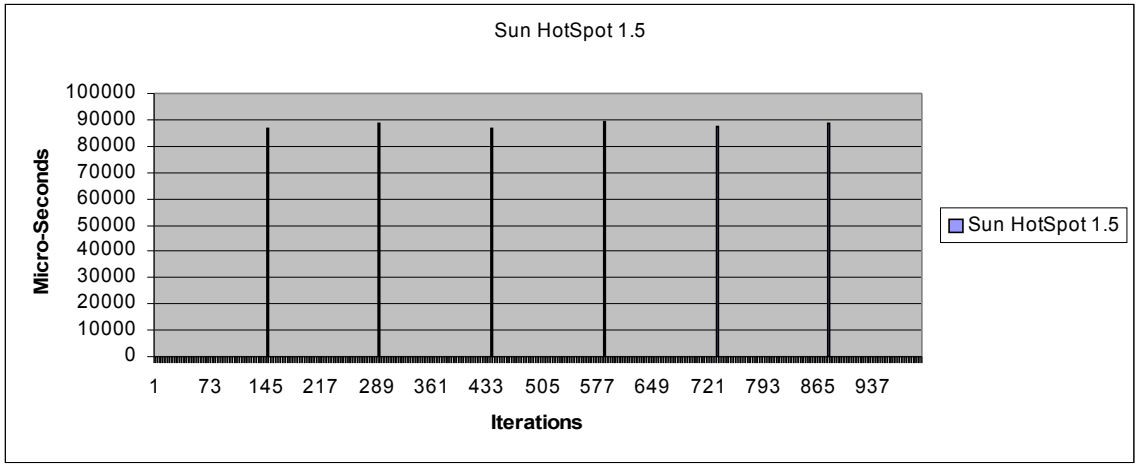
- Writing real-time/safety critical application in the Java™ programming language presents significant challenges. To name a few: Just-In-Time compilation, Garbage Collection, Thread Scheduling, Synchronization Overhead, Lock Queuing Order, Class Initialization, Maximum Interrupt Response Latency, etc.



Typical Execution Time with Just-In-Time Compilation Enabled

Real-Time Virtual Machines for Java Platform

- Most problems can be addressed with new RTSJ Virtual Machines and Real-Time Garbage Collectors.

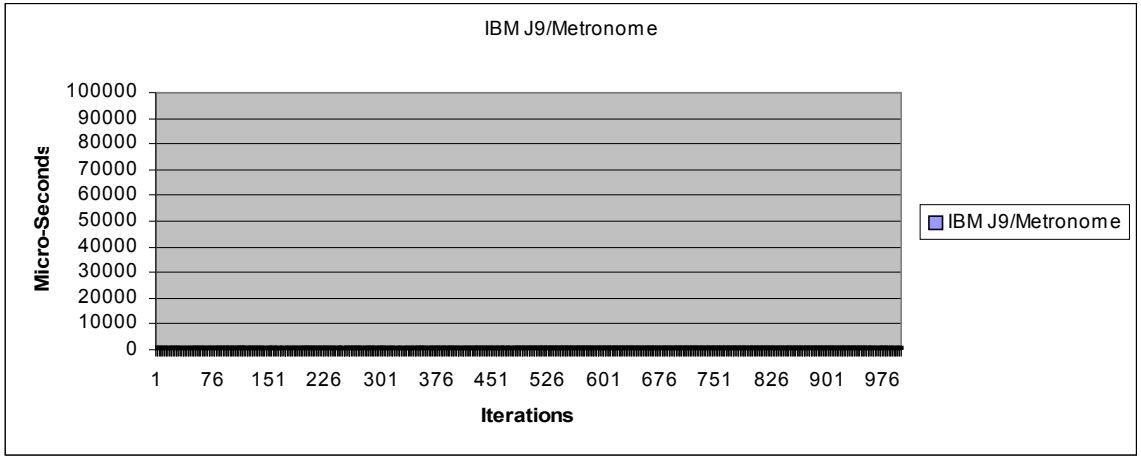


Incremental garbage collection.

Still full-GC occurring from time to time.

Real-Time garbage collection.

GC distributed over time for all objects (short and long-lived).



Still many issues unresolved.

- The standard library is not time-predictable (not even RTSJ-Safe).
- Class initialization is performed at first use and may cascade into hundreds of classes being initialized at an inappropriate time.
- Sharing data with native applications is cumbersome and error-prone (no Struct/Union in Java programming language).

A RTSJ VM is not enough. It needs to be complemented by a real-time library.

Agenda

- Current Challenges
- The Standard Library
- A Real-Time Library
- Context Programming
- Real-Time I/O
- Conclusion

The Standard Java Library.

- Oriented toward throughput (e.g. server applications).
- Time predictability and RTSJ not taken into consideration by the implementation (most of the code written 10 years ago).
- Worst: Its use with RTSJ may result in errors/crashes!
- To this date no time-deterministic or even RTSJ compliant implementation provided by RTSJ Vendors.

Timing Issues

Users may encounter unexpected large delays due to:

- Large arrays being allocated and copied due to internal resizing taking place (e.g. StringBuffer, Vector, ArrayList, etc.)
- Sudden burst of computation (e.g. internal rehashing of hash maps or hash sets).
- Long garbage collection pauses (full GC) due to memory fragmentation when large arrays are suddenly allocated.

Memory Issues

Memory allocation might be performed surreptitiously and cause RTSJ memory clashes. For example:

- A static map instance (allocated in ImmortalMemory) creates new entries resulting in IllegalAssignment errors.

```
//Memory leaks (when entries removed) or IllegalAssignmentError
//(when new entries while in ScopedArea).
static HashMap<Foo, Bar> map = new HashMap<Foo, Bar>();
```

- Objects may be allocated at first use only (lazy initialization) causing further unexpected illegal access errors.

Agenda

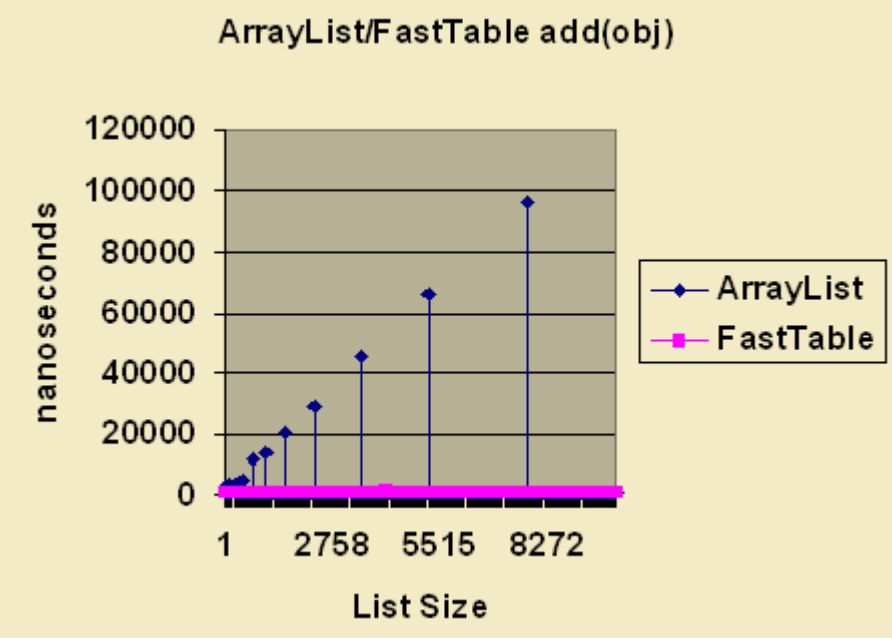
- Current Challenges
- The Standard Library
- A Real-Time Library
- Context Programming
- Real-Time I/O
- Conclusion

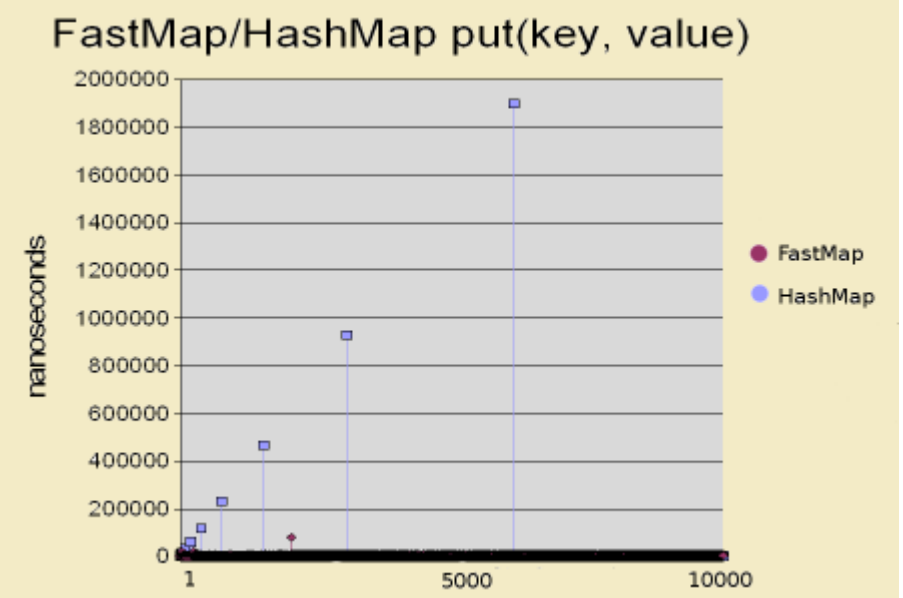
A library for real-time/safety critical applications in the Java platform.

- The RTSJ specification alone is not enough
- To complement the RTSJ effort an open-source project **J**avolution (http://javolution.org) has been created.
- Major RTSJ VM vendors have been invited to participate.
- **J**avolution is already used by a number of large companies Raytheon (Air Traffic Control), Thales, Sun, IBM…

Time-Deterministic Operations

- Operations on **J**avolution classes are highly time-deterministic (in the micro-second range).





RTSJ Compliance

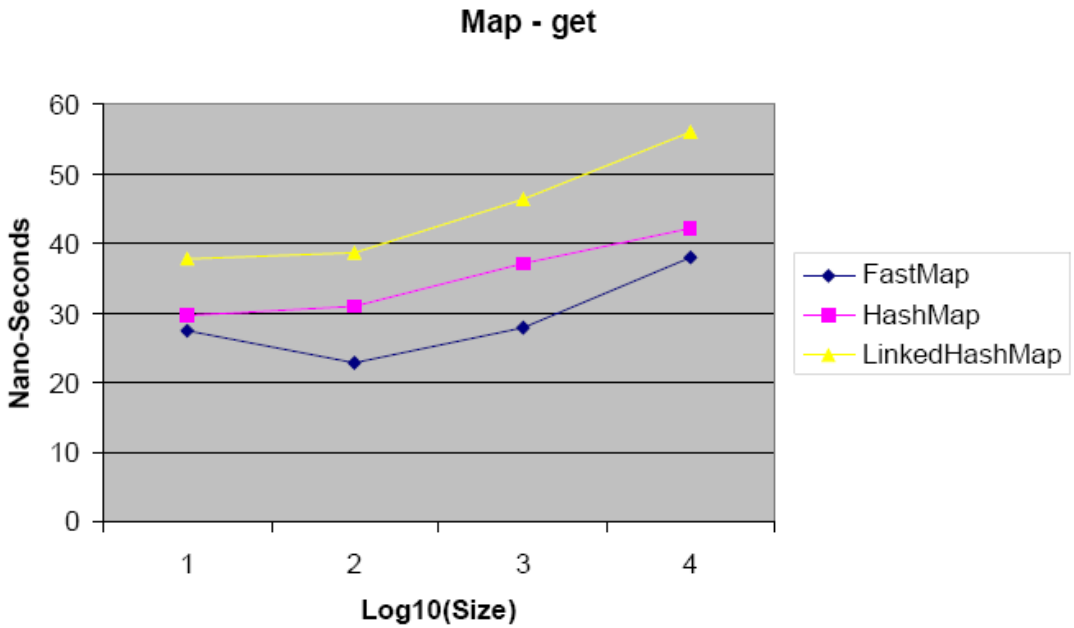
- Javolution classes are RTSJ-Compliant and can safely be allocated in ImmortalMemory
- Javolution classes support real-time resizing (always small increments) and lazy initialization. Any extension part is allocated in the same memory area as the parent object to avoid memory clash.

```
// RTSJ Safe - Removed entries are internally recycled,
// new entries are in ImmortalMemory
static FastMap<Foo,Bar> map = new FastMap<Foo, Bar>();
```

Real-Time AND Real-Fast!

All real-time systems need consistent performance. The RTSJ does not require that a conforming Java platform be unusually fast. But having a poor consistent performance is not better than good performance with some limited fluctuations (often the “worst-case” execution time is actually what matters the most). Here is an example of average time.

Ensuring bounded response-time is of interest to any interactive application (even non real-time).



Agenda

- Current Challenges
- The Standard Library
- A Real-Time Library
- Context Programming
- Real-Time I/O
- Conclusion

Context Programming

- Javolution provides real-time Context to facilitate separation of concerns and achieve higher level of performance and code predictability (e.g. each thread can have its own context based upon criticality, security or performance constraints).
- Here is a list of few predefined contexts:
  - **LocalContext** - To define locally scoped environment settings.
  - **ConcurrentContext** - To take advantage of concurrent algorithms on multi-processors systems.
  - **AllocatorContext** - To control object allocation, e.g. StackContext to allocate on the stack (or RTSJ ScopedMemory).
  - **LogContext** - For thread-based or object-based logging capability, e.g. StandardLog to leverage standard logging capabilities. Note: java.util.logging provides class-based logging (based upon class hierarchy).
  - **PersistentContext** - To achieve persistence across multiple program execution.
  - **SecurityContext** - To address application-level security concerns.
  - **TestContext** - To address varied aspect of testing such as performance and regression.

Case Study - Concurrent Context

- It would be nice when something urgent has to be done that all processors participate in order to terminate the task quickly.
- To this effect, the Javolution library provides a specialized execution context named ConcurrentContext.
- When a thread enters a concurrent context, it may perform concurrent executions at the same thread priority by calling `ConcurrentContext.execute(Runnable)` static method.
- The logic is then executed by a concurrent thread or by the current thread itself if there is no concurrent thread immediately available (the number of concurrent threads is typically limited to the number of processors).

ConcurrentContext - Example

```
ConcurrentContext.enter();
try {
     ConcurrentContext.execute(new Runnable() {...});
     ConcurrentContext.execute(new Runnable() {...});
 } finally {
       ConcurrentContext.exit();
       // Waits for all concurrent threads to complete.
}
```

- ConcurrentContext can be entered recursively (divide and conquer algorithms).

Concurrent Context Success Stories

- Concurrent context has proven to be very efficient. JScience's matrix multiplications for example, are accelerated by a factor 1.99x when concurrency is enabled on a dual-core processor.
- On one government project running on a Sun Fire™ T2000 Server, the execution of lengthy database operations was accelerated 17 times by using concurrent context and only 5 additional lines of code.

Concurrent Context And Server Applications

- Concurrent context has been used with web application servers when some users actions take a long time in order to average the server response time to an acceptable level.
- In such scenario, lengthy operations are performed in a concurrent context and are authorized to use up to half of the processors available.
- This is done through the simple command:

```
ConcurrentContext.setConcurrency(
        (Runtime.getRuntime().availableProcessors()/2)-1);
```

Agenda

- Current Challenges
- The Standard Library
- A Real-Time Library
- Context Programming
- Real-Time I/O
- Conclusion

Real-Time I/O

To support real-time networking, additional classes are provided:

- Struct/Union classes for direct interfacing with application written in C/C++ (e.g. memory sharing or UDP/TCP messages).
- XML Marshalling/Unmarshalling facility (world's fastest according to independent benchmarks)
- Real-Time SAX & StAX XML Reader/Writers using CharSequence instead of String (no garbage generated)

These classes make it possible to write real-time **distributed** applications in the Java programming language!

Javolution Struct/Union

- Javolution provides two base classes to mimic the C struct and union types: Struct and Union
- They follow the same alignment rules, support the same features (e.g. bit fields, byte order, packing) and they make it extremely easy to convert C header files to Java technology-based classes.
- Embedded systems can map Java technology-based objects to physical addresses in order to control hardware devices or communicate through shared memory with external applications
- Struct/Union are wrappers around java.nio.ByteBuffer, tutorials/usages for the Java technology-based NIO package are directly applicable.

C Struct Example

```
struct Date {
        unsigned short year;
        unsigned byte month;
        unsigned byte day;
};

struct Student {
        char         name[64];
        struct Date birth;
        float        grades[10];
        Student*     next;
};
```

Javolution Struct Equivalent

```
public static class Date extends Struct {
    public final Unsigned16 year = new Unsigned16();
    public final Unsigned8 month = new Unsigned8();
    public final Unsigned8 day   = new Unsigned8();
}
public static class Student extends Struct {
    public final Utf8String  name   = new UTF8String(64);
    public final Date        birth  = inner(new Date());
    public final Float32[]   grades = array(new Float32[10]);
    public final Reference32<Student> next =
       new Reference32<Student>();
}
```

Direct Encoding/Decoding

- Struct's members are directly accessible and can be shared with C/C++ native applications.

```
Student student = new Student();
student.name.set("John Doe"); // Null terminated
int age = 2003 - student.birth.year.get();
student.grades[2].set(12.5f);
student = student.next.get();
```

Agenda

- Current Challenges
- The Standard Library
- A Real-Time Library
- Context Programming
- Real-Time I/O
- Conclusion

Conclusion

- Ensuring bounded response time is of interest to any interactive application, even non real-time.
- Safety-critical / Real-Time applications require more than just a low pause virtual machine; a real-time library is essential.
- Javolution has been created in response to the demand for using Java technology in domains it was originally not intended for (Air Traffic Control, Embedded, Real-Time servers).
- Ease of use, good performance and clean design (enforces separation of concerns) may explain its success and use by many reputable companies (Raytheon, Sun, IBM, Excelsior, Lockheed Martin, Thales, BEA, Blockbuster, etc)

For More Information

- JavaWorld™ publication May 2008 “Realistically Real-Time”
- “Fully Time Deterministic Java” by Jean-Marie Dautelle. AIAA Space 2007 Conference Proceedings
- “Validating Java for Real-Time System” by Jean-Marie Dautelle. AIAA Space 2005 Conference proceedings

THANK YOU

Jean-Marie Dautelle, Senior Principal Engineer
Raytheon Company

TS-4797