

JavaOne™
Sun's 2003 Worldwide Java Developer Conference

Designing and Implementing Real-time Control Systems in Java

Al Niessner
Technical Staff
Jet Propulsion Laboratory

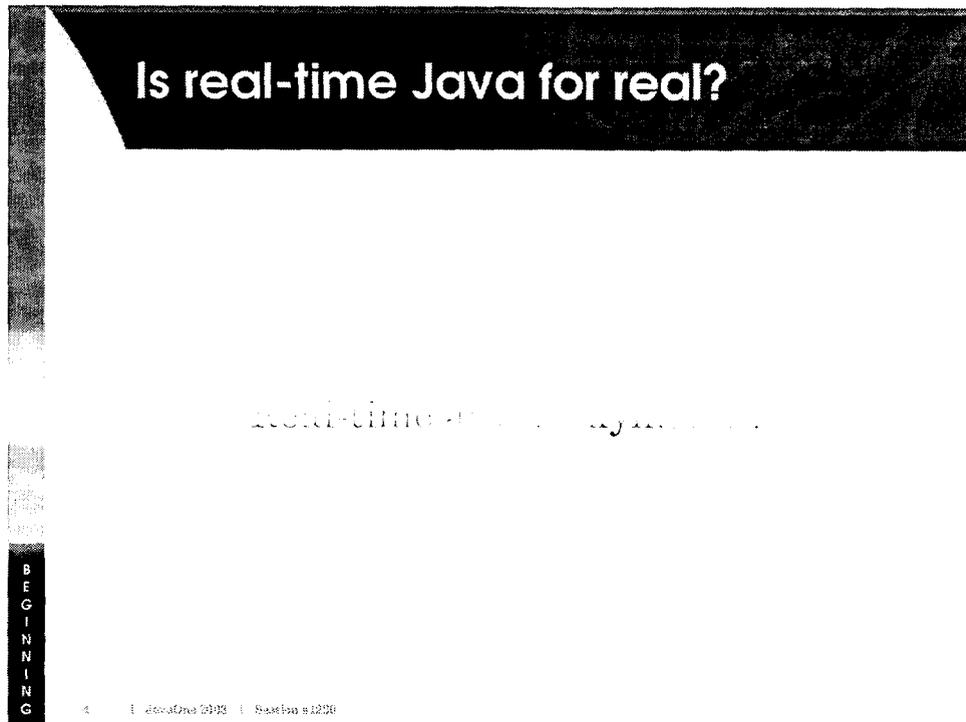
Overall Presentation Goal

Learn how to architect and implement real-time control loop systems using the Real-Time Specification for Java (RTSJ).

BEGINNING

2 | JavaOne 2003 | Session #1220

- Audience should gain the ability to:
 - Separate architecture requirements from real-time/performance requirements
 - Implement a maintainable and extensible control loop architecture



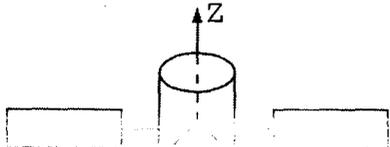
- Story about Java is a web language and could never compete with languages like C/C++ in the embedded, real-time arena.
- Biggest problem with real-time designs is intuitive decision making
- Foreshadow results that it is not an oxymoron.

Control Overview (1/2)

The Equations of State

- **Differential**

$$\sum \vec{F} r + \vec{M}_s = \frac{I}{\dots}$$



$$G(s) = \frac{I}{s^2}$$

- **State Variable**

$$x = Ax + Bu \quad A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}; B = \begin{bmatrix} 0 \\ r \\ I \end{bmatrix}; C = [1 \quad 0]$$

$$y = Cx$$

MIDDLE

- Variable definition table:
 - F = force vector
 - 'r' = radius from center of mass to force
 - M_s = moment from solar pressure
 - I = inertia tensor
 - theta_ddot = acceleration of the body
- Differential equation specifies rigid body.
- Moment from solar pressure is small enough to ignore.
- Frequency model of the plant is double integrator
- State variable model of plant is also double integrator, but not so obvious to the uninitiated.
- The satellite will be considered rigid body for reasons of simplicity. It should be noted that these are text book cases and can be found in almost any book on controls if one wants to understand it better.
- The frequency model of the plant is the easiest to understand in the shortest time. It is simply the La Place transform of the right hand side of the differential equation.

RTSJ Isolation (1/7)

- Scheduler (1/5)

- Periodic Tasks
- One-shot Tasks
- Non-real-time Tasks

• Stateful vs. Stateless Operations
• Priority vs. Deadline

```
public interface TimeEventHandler extends Runnable
{
    public boolean isReady();
}
```

M
I
D
D
L
E

3 | JavaOne 2003 | Session #1220

- The control law was developed in the analog world and the appropriate z-transform will bring it to the digital world. In doing so, it brings along some temporal requirements like jitter and latency limits as well as periodicity
- The system requires that threads be scheduled in three classes of which two are for real-time processing (Periodic and One-shot) and the third (standard Java thread) is for general processing.
- Periodic threads are used to read the sensor and execute the processing loop. Typically the sensor will be read at a rate 10 times faster than the control loop is executed. The data is then filtered in some way to remove the high frequency noise. Hence, the jitter of the periodic thread needs to be at least 100 times smaller than the period of the control loop so that sampling theory still holds. Depending on the K chosen for the compensator, the control loop stability will be intolerant to delays between the sensor and execution of the control loop.
- There are some threads executing that do not care about processing time at all because they need integrated processing power, not instantaneous. Examples are logging, non-essential monitoring, etc.
- Important to note that we extend commonly used interface for making something runnable. We do not try to invent our own.

RTSJ Isolation (3/7)

- Scheduler (3/5)
 - Scheduler Event Connections
 - TimeEventManager

```
public interface TimeEventManager
{
    public void start ();
    public void stop ();

    public boolean doesContain (TimeEventManager id);
    public boolean isEqual (TimeEventManager id);

    public boolean isRoot();
    public TimeEventManager getParentManager();
    public String toString();
}
```

M
I
D
D
L
E

10 | JavaOne 2003 | Session #1220

- The TimeEventManager is basically a thread group. It allows a 'supervisor' or 'manager' to operate on one or many threads in a single call. In our system, we have three primary thread groups:
 - Periodic
 - One-shot
 - Java Threads
- The important calls is the ability to enable/disable a thread. The idea is that it will enable/disable its scheduling and NOT its current execution state. This allows a one-shot to be disabled before it is fired by the thread it is watching.
- In a rate monotonic system, a manager could decide it needs all 100 Hz tasks be disabled to allow the 10 Hz all the processing power. All the 100 Hz could be disabled in a single call until the manager decided otherwise.
- The TimeEventManager is a tree that can be walked either direction and contains a scheduler defined root.

RTSJ Isolation (5/7)

- Scheduler (5/5)

- One-shot Tasks

```
public TimeEventManager schedule(  
    Date          when,  
    String        label,  
    boolean       enabled,  
    DutyCycle     load,  
    Duration      duration,  
    TimeEventHandler timeEventHandler,  
    TimeEventSignal missedTimeEventHandler,  
    TimeEventHandler overrunTimeEventHandler);  
  
public TimeEventManager schedule(  
    Date          when,  
    String        label,  
    boolean       enabled,  
    DutyCycle     load,  
    Duration      duration,  
    TimeEventHandler timeEventHandler,  
    TimeEventHandler overrunTimeEventHandler);
```

M
I
D
D
L
E

12 | JavaOne 2003 | Session #1220

- There are two ways of scheduling for one-shots and the difference is if there is a dead-line that the one-shot must make.
- The first case includes the necessary interface for notifying some entity that the dead-line was missed.
- The second case allows for a duty cycle, but there is no drop dead time that all processing must be completed by.
- In both cases, the thread is called at the specified time (if not disabled at the time of dispatch) and will execute to completion. It will not be rescheduled by the scheduler or any of its components.

RTSJ Isolation (6/7)

- Memory Areas

```
public interface MemoryAllocator
{
    public void enter(Runnable runnable) throws InterruptedException;

    public void enter(Runnable runnable, long timeout)
        throws InterruptedException, TimeoutException;

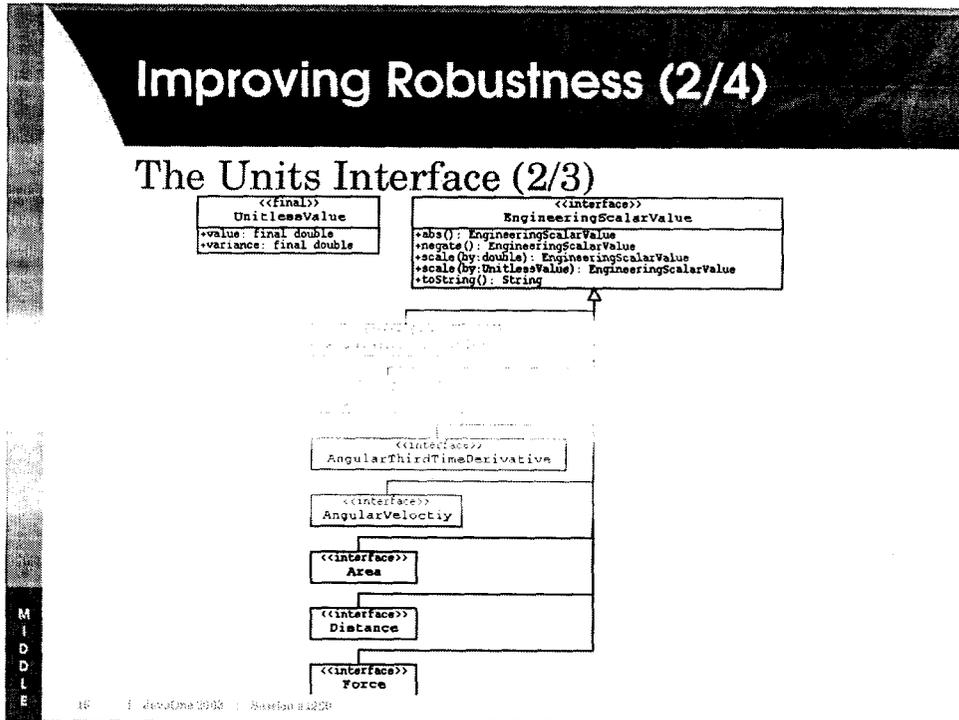
    public Object newArray(Class type, int number)
        throws IllegalAccessException, InstantiationException;
    public Object newInstance(Class type)
        throws IllegalAccessException, InstantiationException;
    public Object newInstance(Constructor c, Object[] args)
        throws IllegalAccessException, InstantiationException;

    public long size();
}
```

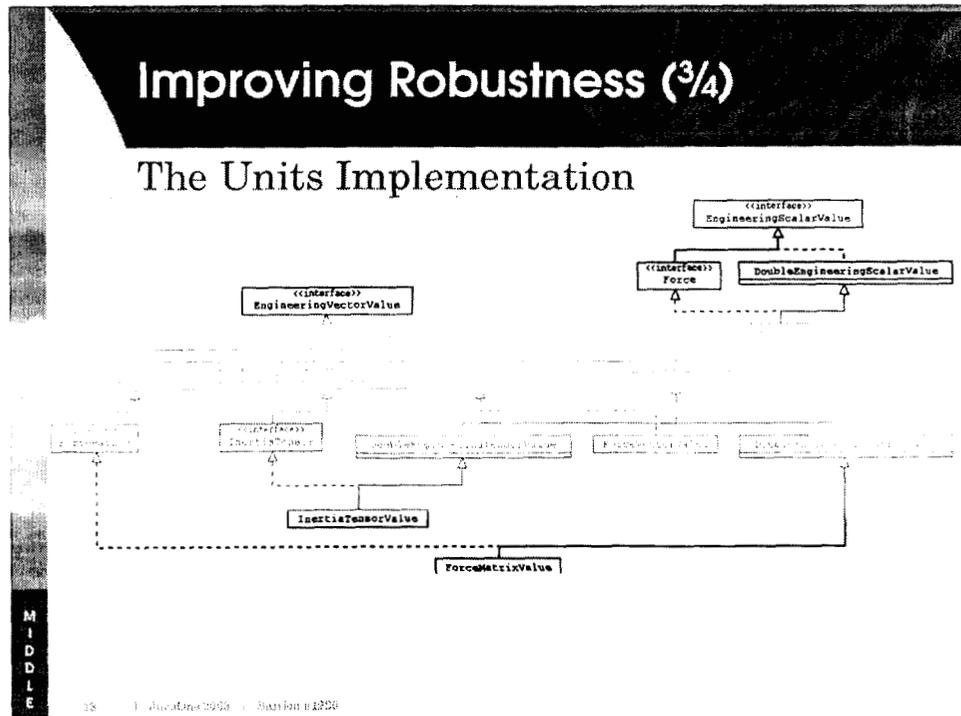
M
I
D
D
L
E

14 | JavaOne 2003 | Session #1220

- The type of memory is an implementation detail and a single interface can be used to represent the gambit. Two side benefits are the pluggable technology and extensibility.
- A thread enters a memory area by calling the enter method with a Runnable. When it is done in the scope, the run() must return.
- The memory can be monitored and checked to see how much memory has been consumed and remains. Nice to prevent out of memory errors
- The total size of the memory area is known as well.



- The class diagram for the interfaces which make up the scalar part of the units. Notice that all of the units are interfaces. This allows the implementer to determine the interaction between them.
- Note that adding a new derived or base unit not currently supported is simply adding a new interface at this layer. The actual work of adding the interaction with the other units depends on the specific implementation of this design.



- Notice the relationships between vectors and scalars – there is none.
- Notice that the inverse mathematical relationship exists between vector, matrix, and tensor. In math, one would say that a vector is a special case of the tensor (rank == 1). In OO, however, a tensor is a child of the vector because it requires more information to uniquely define it.
- The relation between a matrix and vector is the same as between the tensor and vector.
- It then becomes obvious how a self consistent implementation can be built using a standard set of units (mks for instance) and yet they can interact without having to cast from one to another.
- To add a base or derived unit not currently supported requires the addition of the interface from previous slides and then a concrete class whose state is the value and contains some basic implementations of abstract methods from its parent.

Transform Control to Software

- Model basic components of all control loops
 - Sensor
 - Compensator
 - Plant
- Add necessary management
 - Conformity
 - Dynamically changing compensator
 - **Fault tolerance**
 - **Solar panel control**

M
I
D
D
L
E

© | JavaOne 2003 | Session #1220

- All control loops contain three essential components (frequency model). These three components can be mapped directly to interfaces improving the understandability of the software. This can be done for the State Variable model as well. It requires that all control loops be implemented from the same canonical form.
- Some extra management is usually necessary when implementing software modules in a complex system.
 - Typically it is nice to have each subcomponent share a common interface and behavior.
 - Often the compensator changes depending on the mode of the system; e.g., the compensator for detumble mode is different from pointing.
 - In the case of embedded systems, high fault tolerance is required. In these cases, the system must be able to react autonomously to event and exceptions that occur.
 - In our case, the attitude control system also has some particular control over the solar panels.

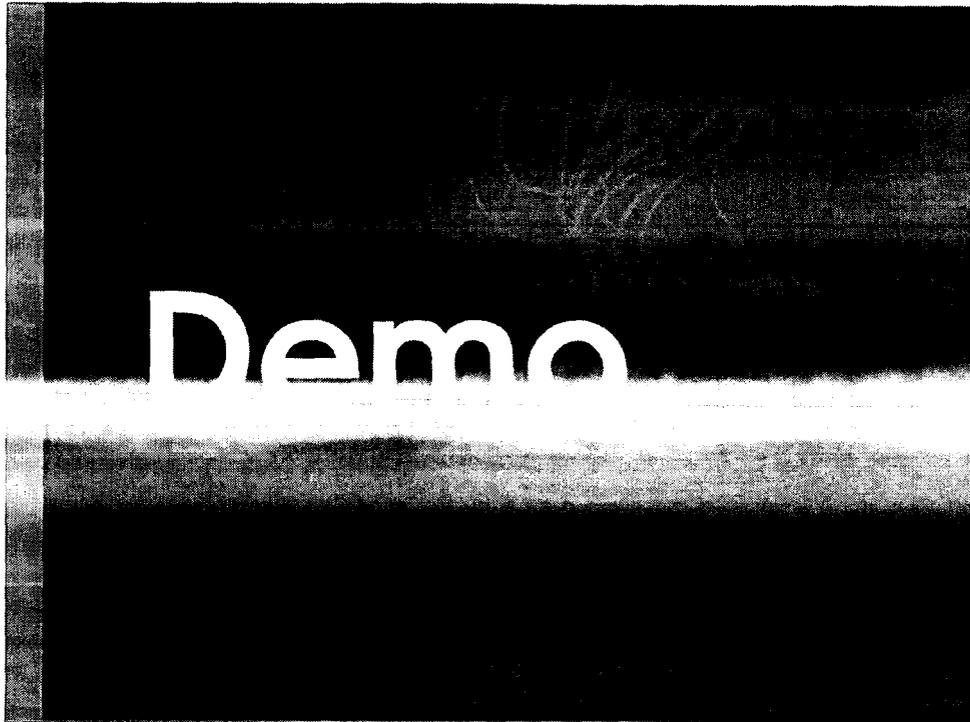
Putting It All Together (1/2)

- Building Control Loops with Units
 - Limits error sources
 - Scale
 - Sign

* Proxy (some manager)

```
periodicId = scheduler.schedule(  
    new Hertz(10),  
    units.zeroTime,  
    "DetumbleLoop",  
    false,  
    scheduler.indeterminate,  
    oneLoopCycle,  
    notifyMeOfAMiss,  
    null);
```

- One major advantage of using units in the compensator is that the error sources become limited. In the case of doing the software for the demo, errors in the control signal were easy to find and remove using analysis tools like Octave. Since all the units are correct, then if the loop did not behave correct, it was a problem with the magnitude of a gain or its sign. Comparing theoretical with software output made it next to trivial to find the typos.
- The control loops are scheduled through proxy – some manager or other. `oneLoopCycle.run()` will be called at 10 Hz with no offset from the epoch. The thread is allocated and ready to run, but is created disabled. In this way, all threads can be allocated at some expensive startup time and then simply enabled/disabled as needed. Currently, we are not using overrun measurements. Lastly, `TimeEventSignal` waiting to do something if the thread misses one of its “rising edges”.



- Two demonstrations: real-time and eye candy
- The first part of the demonstration is reviewing the output of run that operated in real-time.
- The second part is the eye candy (t-ball) where the audience can watch a graphical display of the satellite detumble.

If You Only Remember One Thing...

Real-time is possible in Java and can be more productive and cost-effective than using C/C++.

END

| JavaOne 2003 | Session #1220

- The productivity claims for Java are supported by NIST and can be found at <http://www.itl.nist.gov/div897/ctg/real-time/intro.html>

