MAC
A Run Time monitoring and checking tool

Gursharan Singh

Mohd. Salman Mehmood

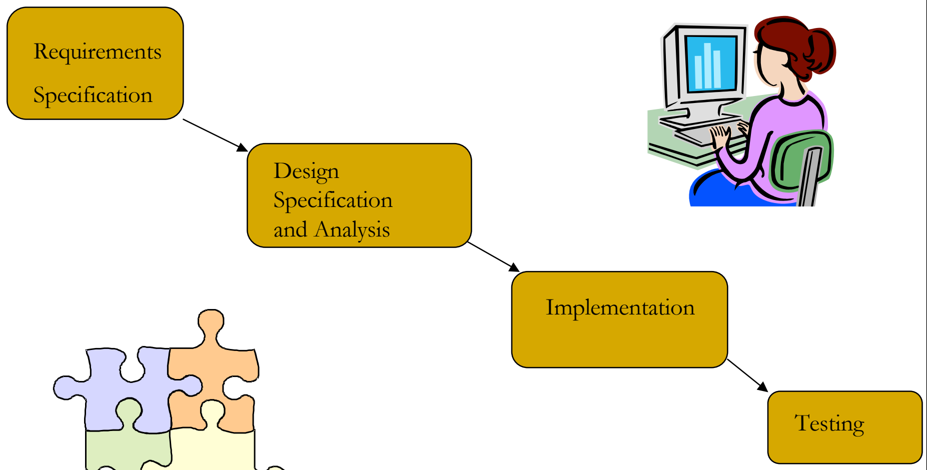
Agenda

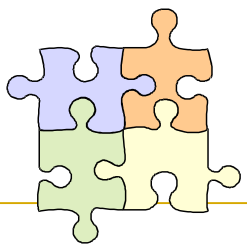
- Motivation
- Software Development
  - Steps
  - Methods
- New Paradigm (Runtime Verification)
- Materializing Runtime Verification (MAC)
- Real Time Java
- Extending MAC
- The road ahead

Motivation

- How is software correctness assured??
  - Verification and Testing !!!!
- How do we go about it ??
  - Understanding development of s/w- flashback

Developing software





Requirements Specification

- What should our software do ?
  - Control Train Gate
- Requirements specified informally-SRS
- Capture requirements formally (Formal specification)

Design Specification and Analysis

- How will the software fulfill the requirements
  - Formal Modeling (UML)
- Analysis
  - Formal Verification (Model Checking)

Implementation

- Implement software in target language
    - Java , C++ , C#
- Test software
    - Black Box Testing
    - Glass Box Testing
    - Unit Testing
    - Stress Test

Software Verification

- Two Phase Verification
    - Design Phase (Formal Verification)
    - Implementation Phase (Testing)

Formal Verification

- Performed at design time
- Great for proving the correctness of the system based on the formal specification
- Explores all execution paths
- Rigorous

What is not good about Formal Verification?

- Checks the model
- Does not verify the implementation
- Not scalable (not feasible for large systems)

- Solution: Verify implementation
  - Software Testing

Software Testing

- Test software based on system use cases
- Test software with different inputs
  - Actual Output==Desired Output (Good)
- Automated testing tools at your dispense
- Testing **assures** that system implementation conforms to the design specification

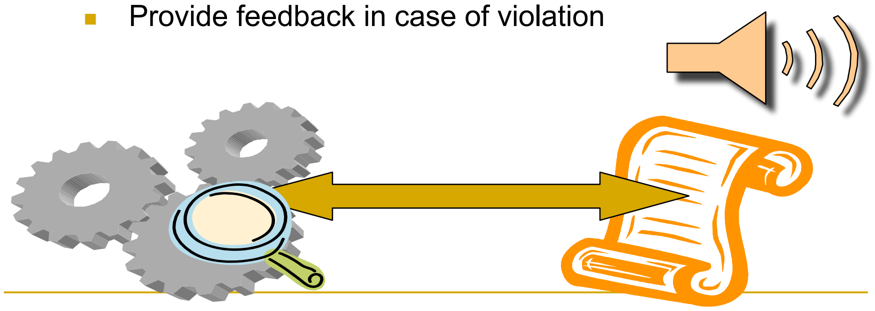
  !!! Fallacy (Cannot test all inputs)

Problems with Testing

- Bugs might not show up during Testing
- Environment specific bug
- Not rigorous
- Not formal

Bridging the Gap

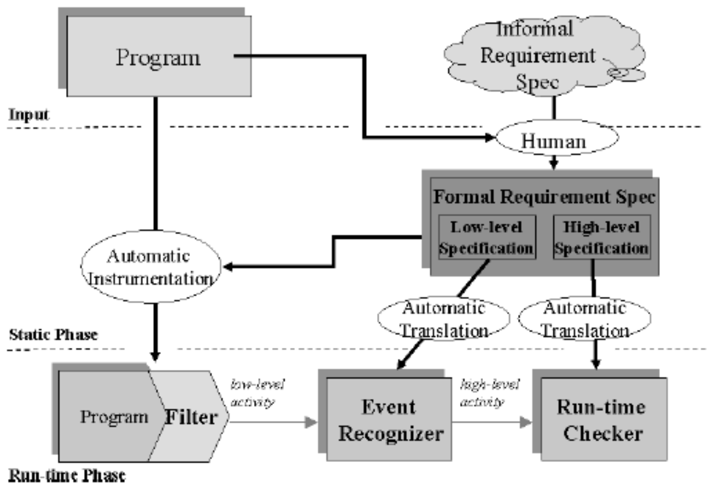
- Runtime Verification
    - Execute implementation
    - Monitor software’s runtime behavior
    - Compare with Formal Specification
    - Provide feedback in case of violation



Enter the MAC

- Monitoring and Checking
    - MAC Vision
        - Verify computational correctness
        - Verify timeliness correctness

Architecture

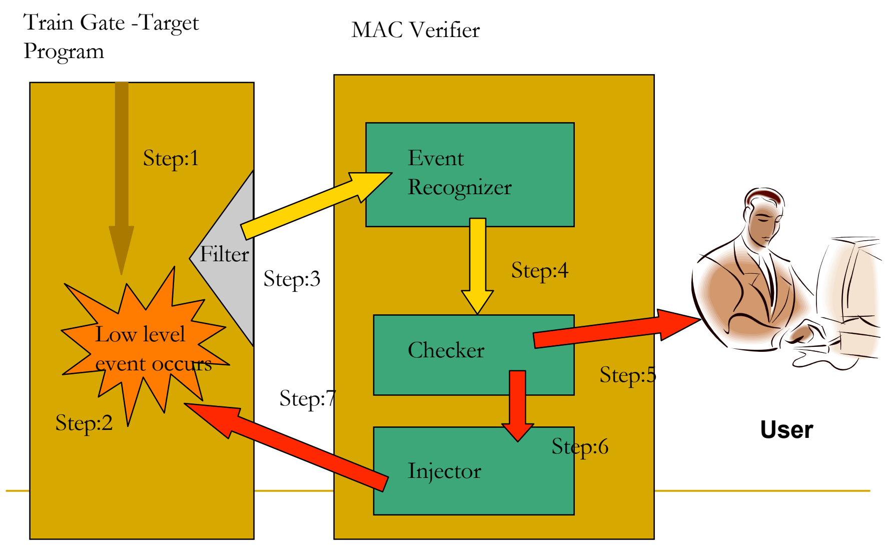


Courtesy: Kim, Lee et al

Static Phase

- Low level formal specs – Implementation dependent
- High level formal specs – Abstracted from implementation, flexibility !
- Code Instrumentation -> Program Filter
- Compiling PEDL (low level)-> Event recognizer
- Compiling MEDL(high level)-> Run time checker

MAC



Instrumentation

- Extending the base software with auxiliary functionality to
  - Monitor program behavior
  - Log significant program events
  - Debugging purposes
  - Performance monitoring
  - Profiling

!!! HOW

Instrumentation (contd..)

- (Static Instrumentation) Inject auxiliary code at compile time
  - Source code
  - Target Byte Code
    - Complicated (Necessary to keep executable consistent)
    +Finer Granularity of observations
- (Dynamic Instrumentation) Inject auxiliary code at runtime
  - On-demand instrumentation (when class loads)
    Example Sun One Application Server for profiling purposes

Instrumentation (contd..)

- Manual Instrumentation
  - Hand assembled by user
    + Easy to pin point monitoring locations
    - Result in incomplete instrumentation
- Automatic Instrumentation
  - Instrumentation Bot e.g JTrek
    + Complete (Captures all significant events), Why?
      - Mechanical
    - Difficult to define HL events based on LL state information

Train Crossing Demo

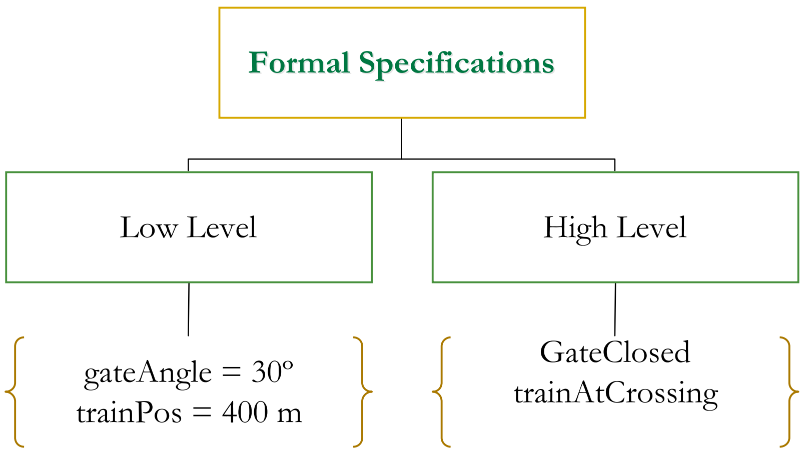
Example – train crossing

- Gate to be closed when train reaches crossing
- Can only open when train has crossed
- If (!GateClosed && trainAtCrossing) then

```
    raise alarm
Else
    continue
```

- What is GateClosed? trainAtCrossing?

System Specifications

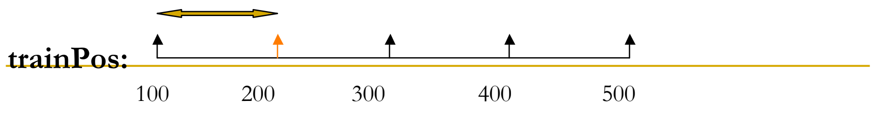


Events

- Instantaneous happenings in the system
- E.g. : Entry/Exit into a method, update of a variable

Conditions

- Information that holds for some time
- E.g. : trainPos < 200



Syntax

- Events : <E>
  - <E> = e | start (C) | end (C) | (E && E) | (E||E) | E when C
- Conditions : <C>
  - <C> = c | defined (C) | (E , E) | ! C | (C && C)| (C||C) | (C => C)

PEDL – Primitive Event Definition Language

- Used to define low level specs
- Dependent on target program language
- Instrumentation should not inflict resource overheads – CPU
- Functional behavior of the program should not be affected
- Fast recognition of events

Java PEDL – train crossing example

- MonScr GateCrossing
    /* Export section */
    export event startTrain, gateHeight;

  Monobj int Train.run;   //monitored objects
  Monobj int Gate.Roll().gateHeight;

   event startTrain = startM (Train.run());   // event def
   event gateHeight = update (Gate.Roll().gateHeight);

  end

MEDL

- ReqSpec GateCrossing
    import event startTrain, gateHeight ;

  var int angle;

  Var bool trainRunning;

  Condition gateClosed = (trainRunning && angle >= 0 &&
    angle <= PermissibleError)

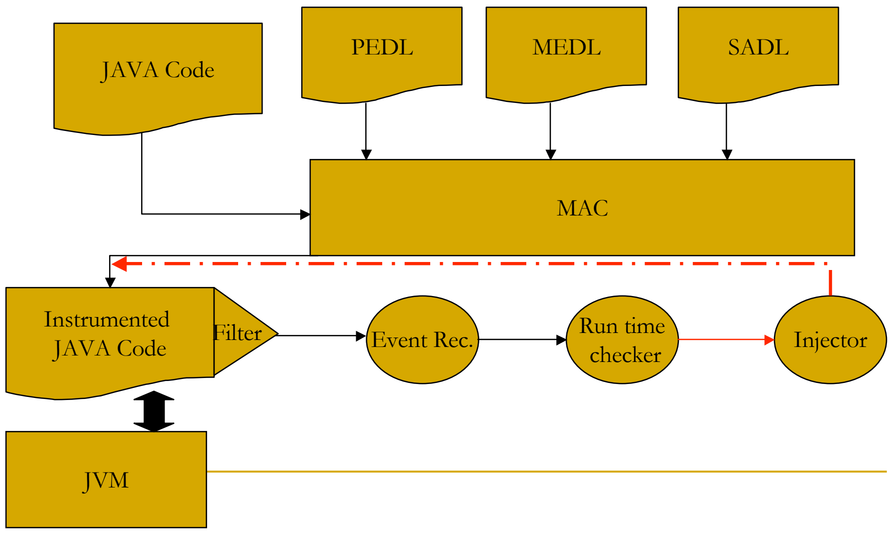
  Alarm possibleAccident = start (!gateClosed);

  startTain -> { trainRunning = 1; angle = 90;}

Java Implementation Specifics

- Objects not monitored directly, why?
  - References inside objects difficult to monitor, why?
    - Any update requires passing the whole object graph to run time checker
    - Recursive tracking
- Monitoring only fields serves the purpose – change in object -> change in a field

The Java prototype



Faults and Failures

- According to theory of fault tolerance -
  - Fault : an unwanted change in a system
  - Failure : result of a propagating fault
- So what did we just do ?
  - We detected the fault
  - Now we need to stop this from developing into a failure –> STEERING!

Steering

- Detect a violation, inject correcting code
  - E.g. Train.speed = 100 at crossing
    Inject code : Train.speed = 40 before call gate.close()
- Run time checker invokes “steering actions”
- Steering actions specified by the user – static, done during instrumentation

SADL – Steering Action Definition Language

- Since steering is done in the executing code, SADL tied to source language
- Static Code injection – putting in the correcting code
- Works by setting flag – concept can be understood as:
  - if(flag) then

```
    action
else
    normal execution
```

Contd..

SADL Script

- Steering script gateCrossing

```
  // steering entities
  steered objects boolean Gate: gateUp;
  void Gate: rollUp( );

//steering actions
Steering action holdGate =
    { Gate: gateUp = false;} before call Gate:
  RollUp( )
```

SADL

- Action Invocation in MEDL
  - possibleAccident -> { holdGate };
- You need to know what to monitor
- Only Real Code monitorable – what about virtual machine specific code
- What about side effects? Heap, GC?

Shortfalls

1. Long code, more to-be-monitored vars, more SADL, overhead on program memory-embedded devices?
2. Side effects like heap, GC and other run time system dependent features
3. If fault and failure time difference not much
4. What if all fields are references to objects?

But, they don't really hurt ☺

- 1 & 4 can be handled

Long code, more to-be-monitored vars, more SADL, overhead on program memory-embedded devices?

What if all fields are references to objects?

A= a+1;

B= a+1;

Why monitor A and B both?

Smart Programming and designing

Extending MAC

- Existing system is based on "Event Driven" evaluation
- Verifies only computation correctness
  - Example . The gate opens when the train has crossed
- For verifying timeliness correctness we introduce notion of "Time Driven" evaluation

Choose RT-Java

- RT Java extends standard java with real time aspects

  + Reuse existing code base

  - Architectural changes to MAC compiler

  +No need to learn new language

!!! Lets have a look at Real Time Java !!!

A Brief Primer on RT Java

Why do we need RT Java?

- Standard Java is unpredictable in terms on thread schedulability
- Weak concurrency model
- Lacks support for Real Time Systems
- Undeterministic Garbage Collection

Enhancements to RTSJ

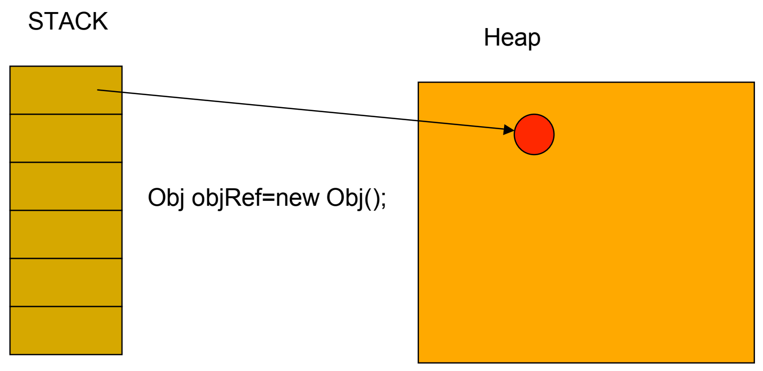
- Memory Management
- Time Values and Clocks
- Schedulability Objects
- Real Time Threads
- Async Event Handling and Timers

Memory Management

- Memory partitioned into different areas
- Instruct jvm to place data types in different areas
- Improve performance
- Improve predictability

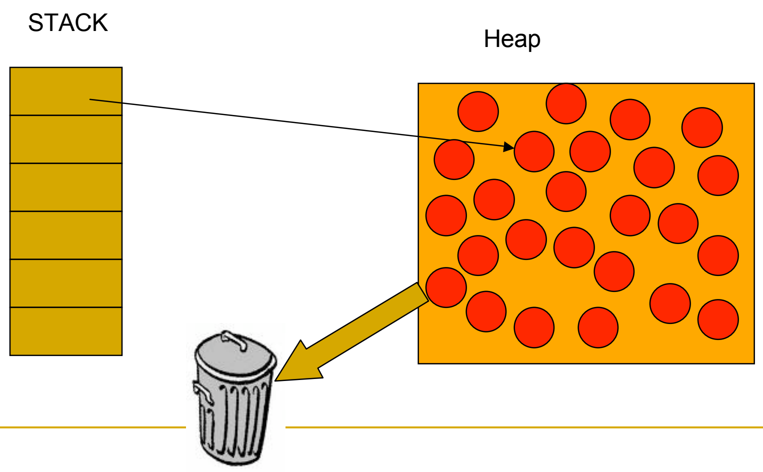
!! Step Back and look at standard java memory

Heaps and Stacks



Garbage Collection

Reclaim memory when Heap becomes full ,other memory reclaiming schemes



Memory Areas

- Heap Areas
  - Same as standard java
- Immortal Memory
  - Objects are not reclaimed
- Scoped Memory Area
  - LTMemory Area
  - VTMemory Area

Time Values and Clock

- High resolution Time
  - NanoSecond granularity
  - Absolute Time (relative to some epoch)
  - Relative Time (relative to some local clock)
  - Rational Time (relative + associated frequency)
- Timers
  - Countdown clocks

Schedulable Objects

- Standard Java lacks scheduling predictability
  - Cant guarantee if high priority thread is the one that will be executing at any time
  - Depends on host OS to support threads
- RTSJ introduces notion of schedulable objects

Schedulable Objects

- Implement Schedulable interface
- Indicate specific release requirement
  - Periodic (regular)
  - Aperiodic (random)
  - Sporadic (irregular with minimal inter-release times
- Indicate memory requirements
- Scheduling requirements

Meeting Deadlines

- Obligation of the scheduler
- Predict whether a set of application objects will meet their deadlines
  - RTSJ provides hooks to support on-line analysis
- Notification in case an application object
  - misses deadline
  - consumed more resources than specified
  - released more often

Real Time Threads

- Is a schedulable object
- Extends the standard Java Thread
- Uses
  - Release Parameters
  - Memory Parameters
  - Scheduling Parameters

- NoHeapRealTimeThread <extends>> RT Thread
  Does not create or reference objects on the heap
  Execution independent of garbage collector

Async Event Handlers

- Use extra thread for async event handling (inefficient)
- Multiplex events onto a pool of threads
- Event Handlers need to respond within deadlines
- Events Handlers are schedulable entities
- Bound Event Handlers (dedicated RT Thread)

Now back to RT MAC

RT Java MAC

- MAC porting to Real Time Java to achieve timeliness correctness verification
- A single Java program that includes
  - Target Program+Filter
  - Event Recognizer
  - Checker
- Unlike MAC which had three separate java programs

Time bounded property

- [e1,e2)<=d
  - Semantics
    - Event e1 occurs , and the condition is valid if e2 occurs within deadline d ,else violation
    - Extend MEDL and PEDL with Temporal operators

RT-MAC Components

- Target Program + Filter
  - Implemented as Real Time Thread
  - Schedulable object in RT Java
    - Specify different parameters for the RT thread
      - ReleaseParameters
      - MemoryParameters
      - Scheduling Parameter
- Event Recognizer & Checker
  - Implemented as Async Event Handlers
- Communication across components via Async Event and Shared Variables

RT-MAC Mechanics

- Target Program+Filter
  - Instrumentation code fires a lowLevelAsync Event
  - Async Event handled by Event Recognizer
- Event Recognizer
  - Transform low level information into events and conditions
  - Fires abstractAsyncEvent which is handled by Checker (AsyncEventHandler)
- Checker
  - Evaluates events and conditions to determine computational and timeliness correctness

When do we perform an evaluation

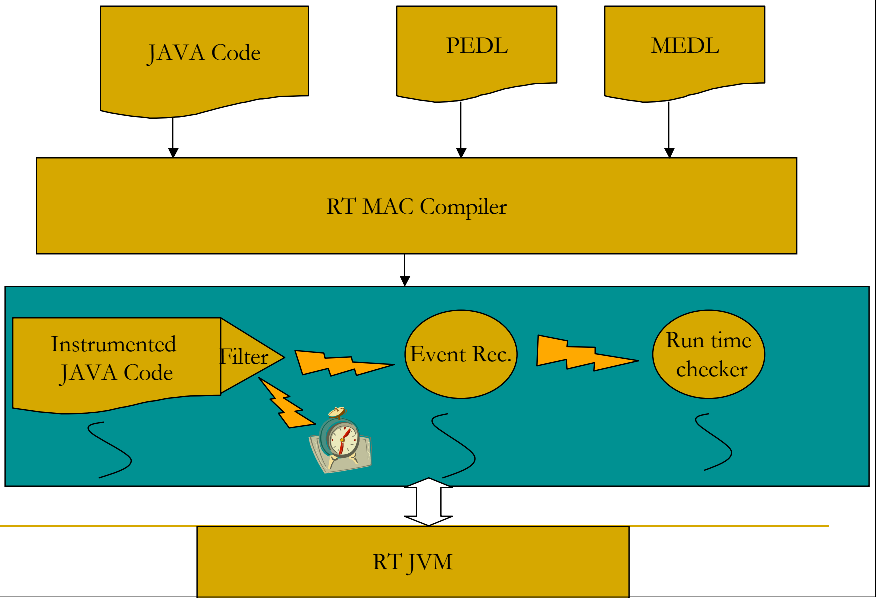
- Periodically (HEART BEAT)

  +Easy (implemented using a periodic timer)
  - Evaluate whenever timer expires

  - Redundant and Inefficient

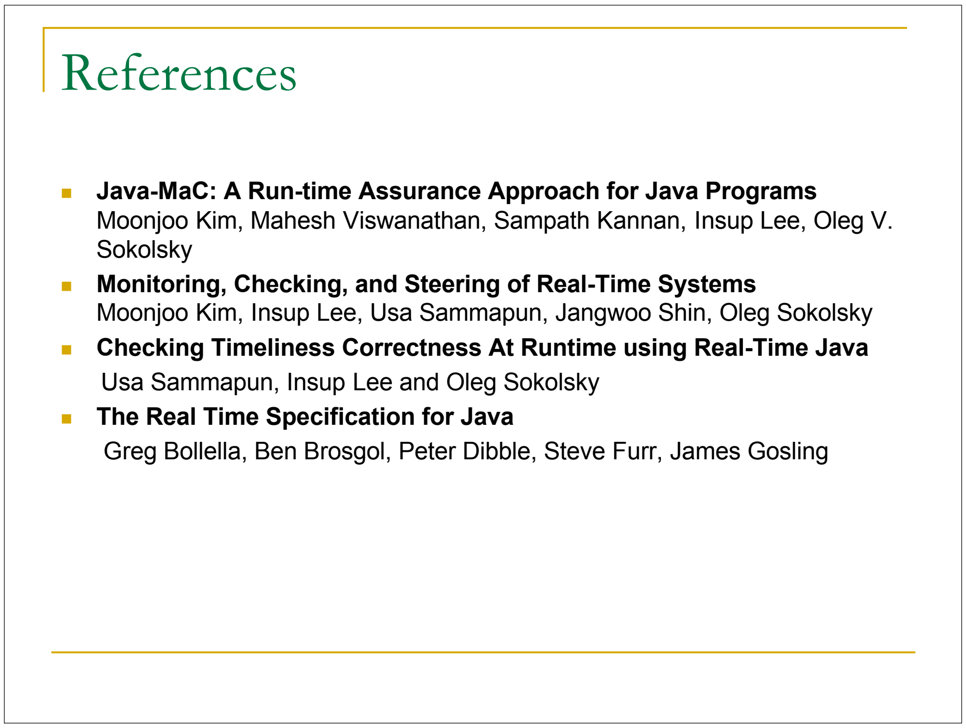
  -Need to determine optimal time steps
  - Time step too small (increased overhead)
  - Time step too big (might miss deadline,evaluation not done as soon as possible)
- Occurrence of Event (Time Out) [e1,e2)<=d
  - Set timer when event e1 occurs
  - Event e2 occurs before timer expires (Good)
  - Else violation detected

RT MAC Architecture



Future Work

- RT implementation
- Soft Real Time systems – add counters
- Heuristics to detect errors
- Playing with the VM – monitor some of the VM threads
- Avoiding nested failures, one failure, another upcoming? Yes-> recover.



References

- **Java-MaC: A Run-time Assurance Approach for Java Programs**
  Moonjoo Kim, Mahesh Viswanathan, Sampath Kannan, Insup Lee, Oleg V. Sokolsky
- **Monitoring, Checking, and Steering of Real-Time Systems**
  Moonjoo Kim, Insup Lee, Usa Sammapun, Jangwoo Shin, Oleg Sokolsky
- **Checking Timeliness Correctness At Runtime using Real-Time Java**
  Usa Sammapun, Insup Lee and Oleg Sokolsky
- **The Real Time Specification for Java**
  Greg Bollella, Ben Brosgol, Peter Dibble, Steve Furr, James Gosling