Debugging Distributed Object Applications with the Eclipse Platform

Giuliano Mega and Fabio Kon

Technology eXchange

Context

- *Distributed Systems*
  - *Notoriously difficult to build without appropriate assistance.*
  - *First ones were based on message passing mechanisms.*
  - *Parallel programming libraries (such as MPI) made implementation a bit easier.*
- *Nowadays*
  - *Object-Oriented Middleware.*
  - *Designed primarily for distributed (not parallel) systems.*
  - *Also a design framework.*
  - *Distributed Object Systems (**DOS**es).*
  - *Been around for a while.*

Motivation

- *Debugging*
  - *Older systems: less abstraction meant simpler debuggers.*
    - *Not much difference between the code manipulated by the user and what actually got executed.*
  - *Abstraction frameworks introduce code.*
    - *not meant to be seen by the user at development time.*
    - *little care is taken at runtime.*
  - *Result: large discrepancies*
    - *Middleware: development made easier; or*
    - *Middleware: debugging made harder?*

Motivation (cont.)

- *Related tasks*
  - *Testing*
    - *Debuggers help fix errors.*
    - *Testing is paramount for finding them.*
  - *Setting up test scenarios is also a part of the debugging process.*
    - *Launching remote processes.*
    - *Simulating failure and observing system behavior.*
  - *Collecting remote data.*
  - *A basis for automated testing.*

Basics

- *Debug modes*
  - *Live debugging*
    - *Passive (no interaction)*
    - *Active (interaction)*
    - *High impact on performance (in general), scales poorly.*
    - *Accurate, “live” information.*
  - *Offline (“dead”) debugging*
    - *Can only be passive.*
    - *Low overhead (comparing to live).*
    - *Only available at the end of the execution.*
    - *Ideal for postmortem analysis.*
  - *Tradeoff*

Basics (cont.)

- *Causality*
    - *Becomes an issue when distributed systems are concerned.*
    - *Wrong causality analysis means useless trace information or wrong state displays for live debuggers.*
- *What causality-related information is there to capture?*
    - *Caller/callee relationship [1];*
    - *thread parent/child relationship [2];*
    - *dynamic dependencies [3];*
    - *properties on cuts;*
    - *etc.*

Our approach

- *Simplest idea possible*
  - *OO Middleware allows developers to think of their Distributed Systems as if they were multithreaded, Object-Oriented systems.*
  - *We want to:*
    - *preserve that illusion at debug time.*
    - *level interactivity with what's provided by today's source-level debuggers.*
  - *Involves capturing key points of system execution and displaying them to the user, hiding middleware-related code execution (debug time complexity-hiding).*

Our approach (cont.)

- *Synchronous-call mechanisms*
  - *Induce the distributed thread concept.*
    - *Threads that span multiple machines.*
    - *Core element for causal relations in **DOS**es.*
- *Distributed thread visualization*
  - *Our hypothesis*
    - *Expected extension for symbolic debuggers;*
    - *could lead to valuable insight;*
    - *possible source of information for other algorithms.*

Our approach (cont.)

- *However*
  - *Arguably useful [4]*
    - *Maybe it doesn’t lead to valuable insight after all.*
  - *Scales poorly*
    - *Could be an issue for some applications.*
  - *Difficult to implement*
    - *Little support from runtime environments.*
  - *Could be unworkable in some situations*
    - *Too much interference.*
    - *The “timeout” issue.*

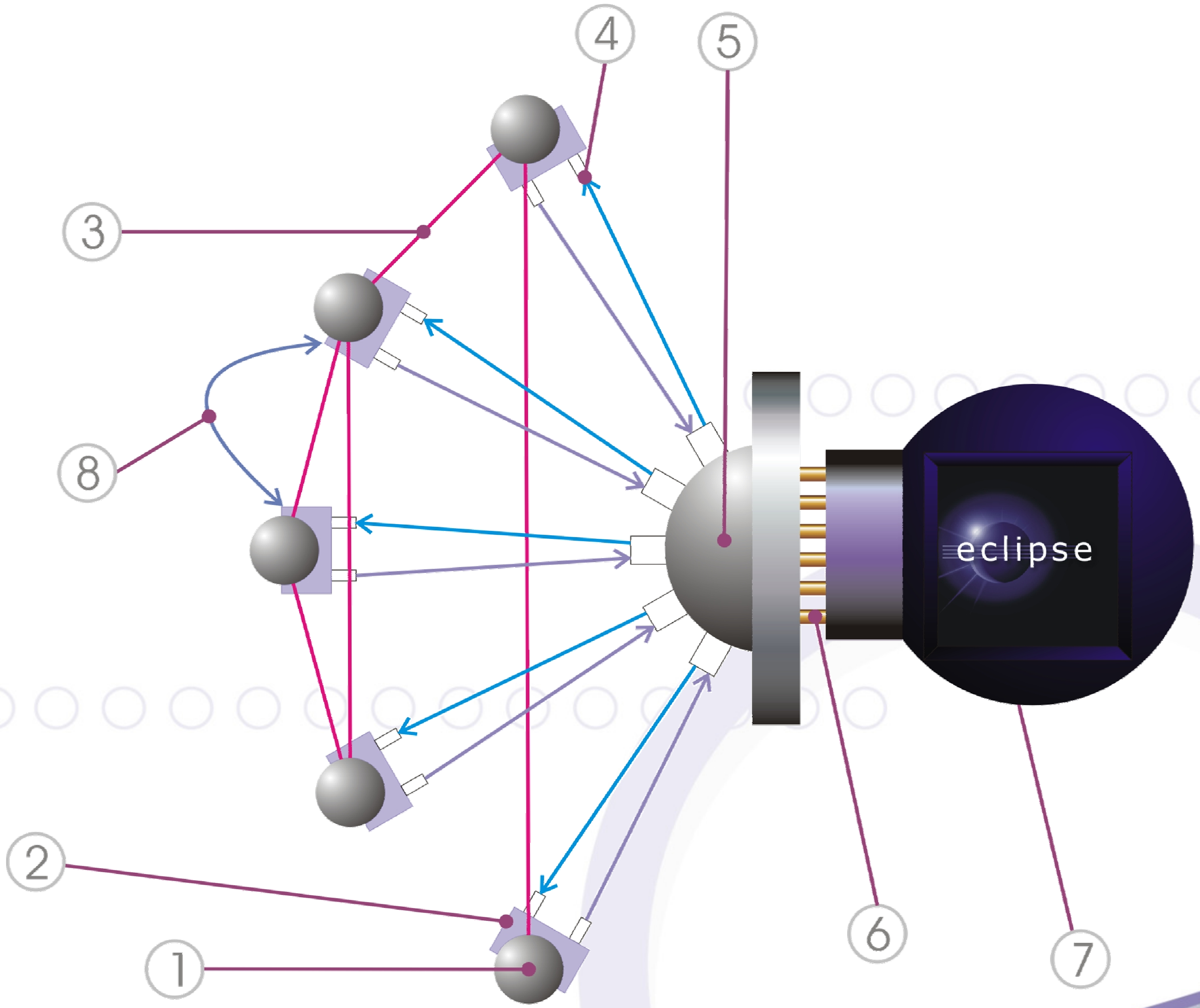
Initial implementation

- *Targeted at Java/CORBA environments.*
- *Allows (*will be available shortly):*
  - *dynamic distributed thread tracking and visualization;*
  - *fine grained control over the flow of execution (stopping, resuming and inspecting distributed threads);*
  - *arbitrary state inspection;*
  - *launching/killing remote processes;*
  - *on-the-fly data visualization;*
  - *stable property detection*.*
- *Would be perfect if*
  - *could replay execution and;*
  - *detect unstable predicates.*

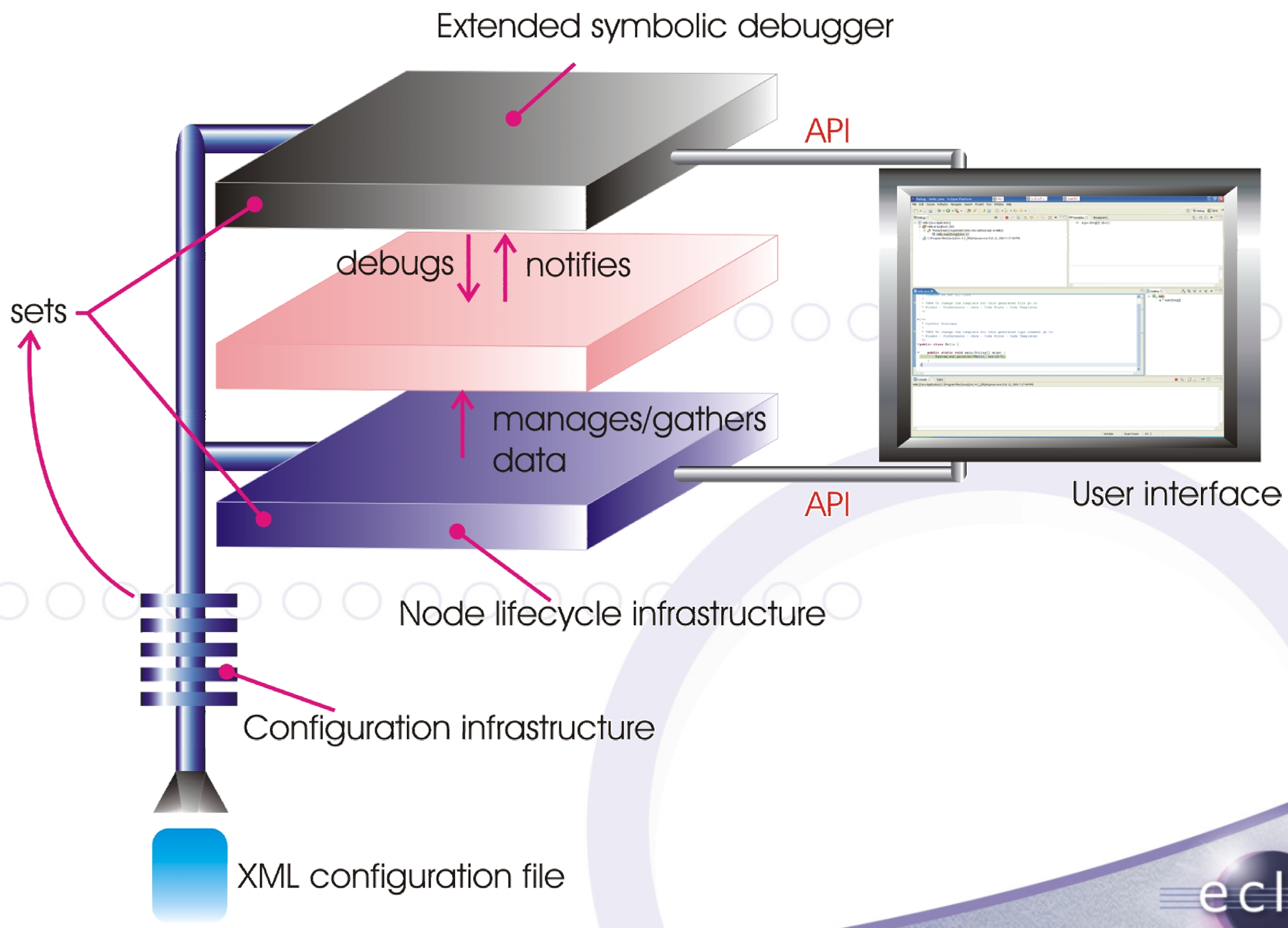
Eclipse

- *Provides the model*
  - *Sophisticated interface and interaction set*
- *Perfect environment for such a tool*
  - *Extensive support from the framework;*
  - *open source implementation;*
  - *widely adopted.*
- *Phase 1:*
  - *Model GUI extensions for accommodating distributed threads;*
  - *extensions for controlling remote applications and*
  - *visual aids (call graphs, dynamic distributed thread visualization).*

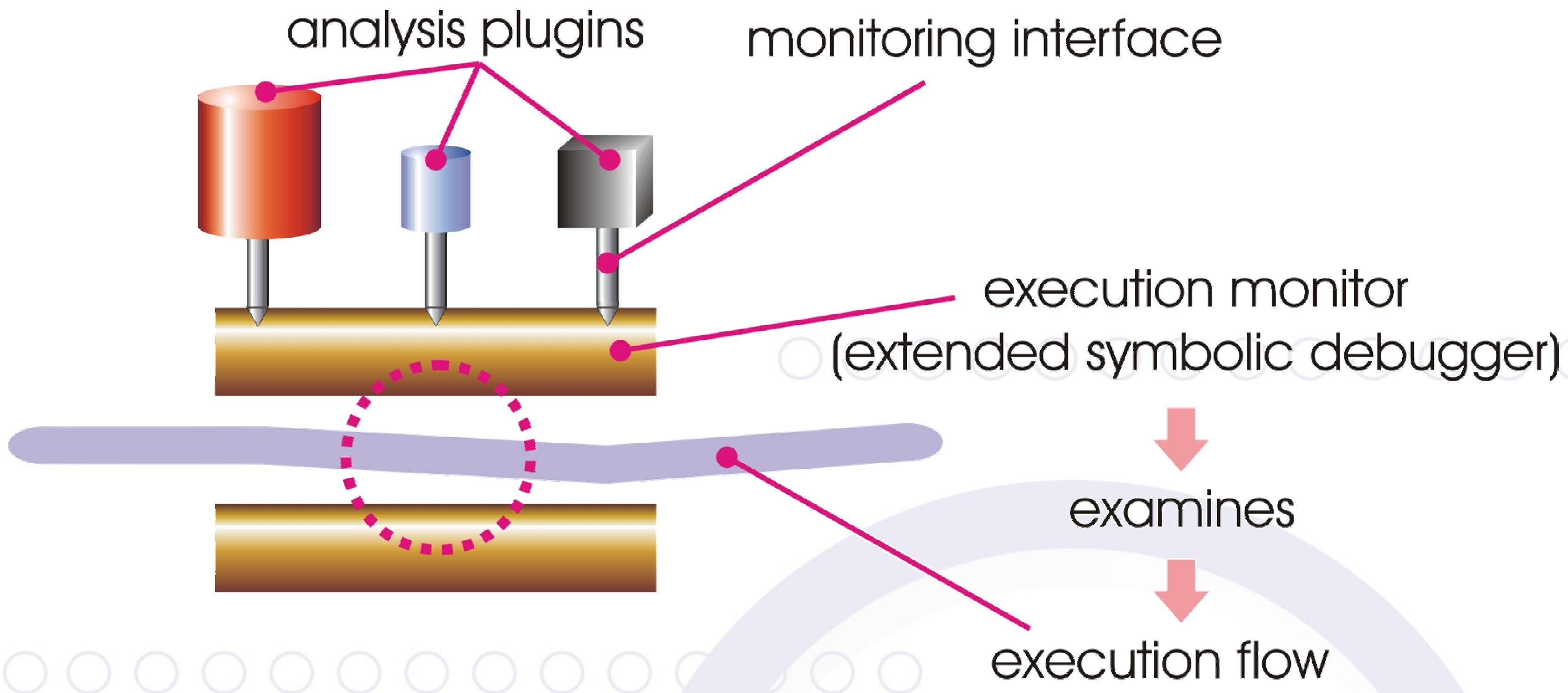
Architecture



Architecture (application)


Extended symbolic debugger
API
debugs
notifies
sets
manages/gathers data
User interface
API
Node lifecycle infrastructure
Configuration infrastructure
XML configuration file

Architecture (data analysis)

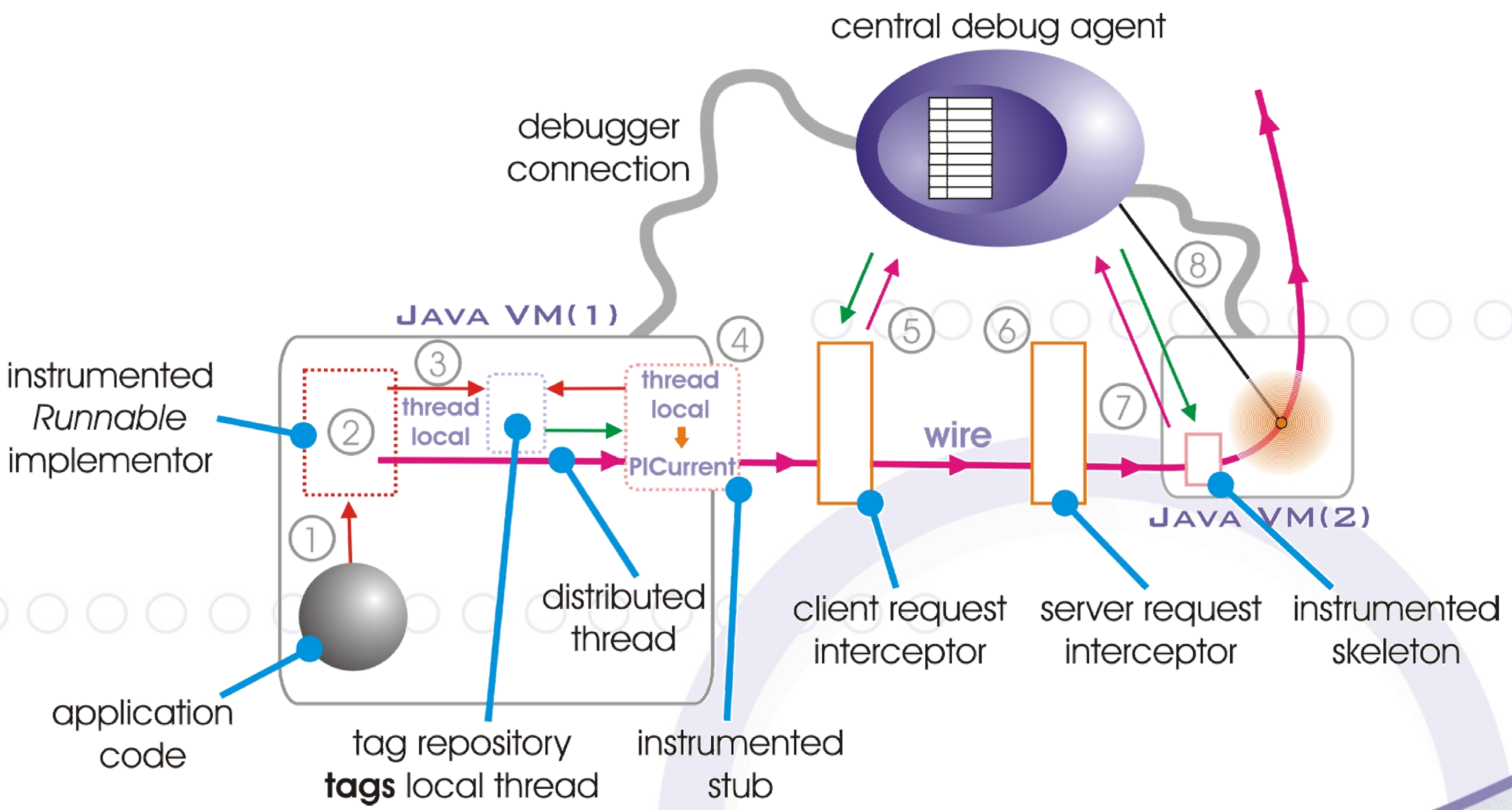


- This is the planned design for the data processing architecture.
- Doesn’t reflect what’s implemented.

Tracking distributed threads

- *Involves*
  - *Mapping local threads to distributed threads;*
  - *assembling "virtual stacks";*
  - *knowing which thread is where and when.*
- *Virtual stacks*
  - *Extended call stack concept;*
  - *partially describes the causal relations inside a single call chain;*
  - *allows arbitrary state inspection of running distributed object application.*

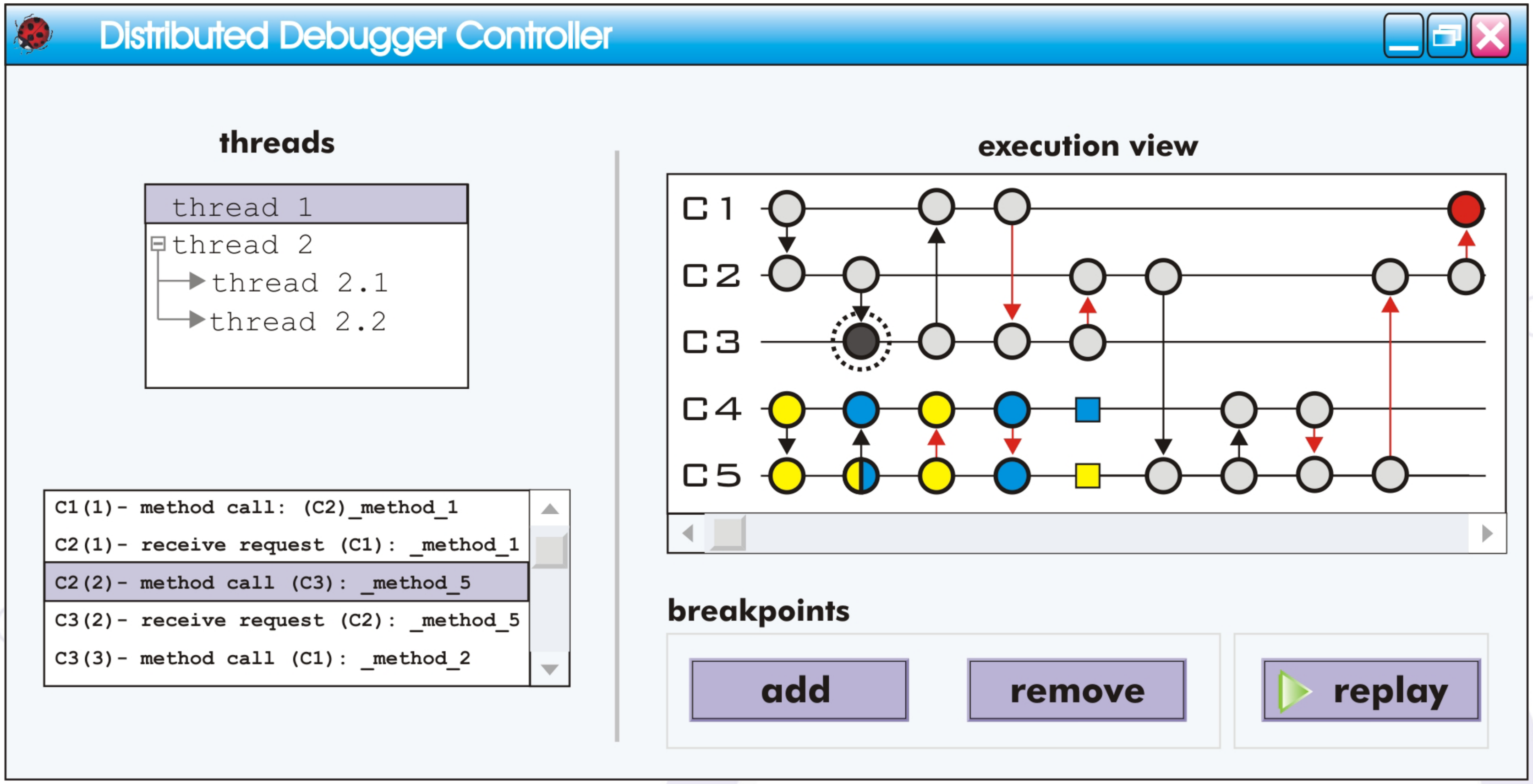
Distributed thread tracker



Limitations/considerations

- *Works only with synchronous-call models;*
- *timeouts must be disabled for state inspections to work;*
- *currently tied to Java/JDI;*
- *could produce too much overhead;*
- *could fail miserably with some ORB thread handling schemes;*
- *limited support for remote process management.*
- *We *must* improve laziness of distributed thread tracking.*

Rough "screenshot"



```
C1(1)- method call: (C2)_method_1
C2(1)- receive request (C1): _method_1
C2(2)- method call (C3): _method_5
C3(2)- receive request (C2): _method_5
C3(3)- method call (C1): _method_2
```

Availability

- *More information (including source code) can be found at the project web site:*
  http://eclipse.ime.usp.br/projects/DistributedDebugging
- *Thank you!*

Questions?