http://bitser.net/isolate-interest

What Java Isolation can do for you (someday, but not in Tiger)

An introduction to Java Specification Request JSR-121

Pete Soper
pete.soper@sun.com

Portions of this material previously presented by:

Doug Lea
State U. of NY/Oswego
dl@cs.oswego.edu

Miles Sabin
miles@milessabin.com

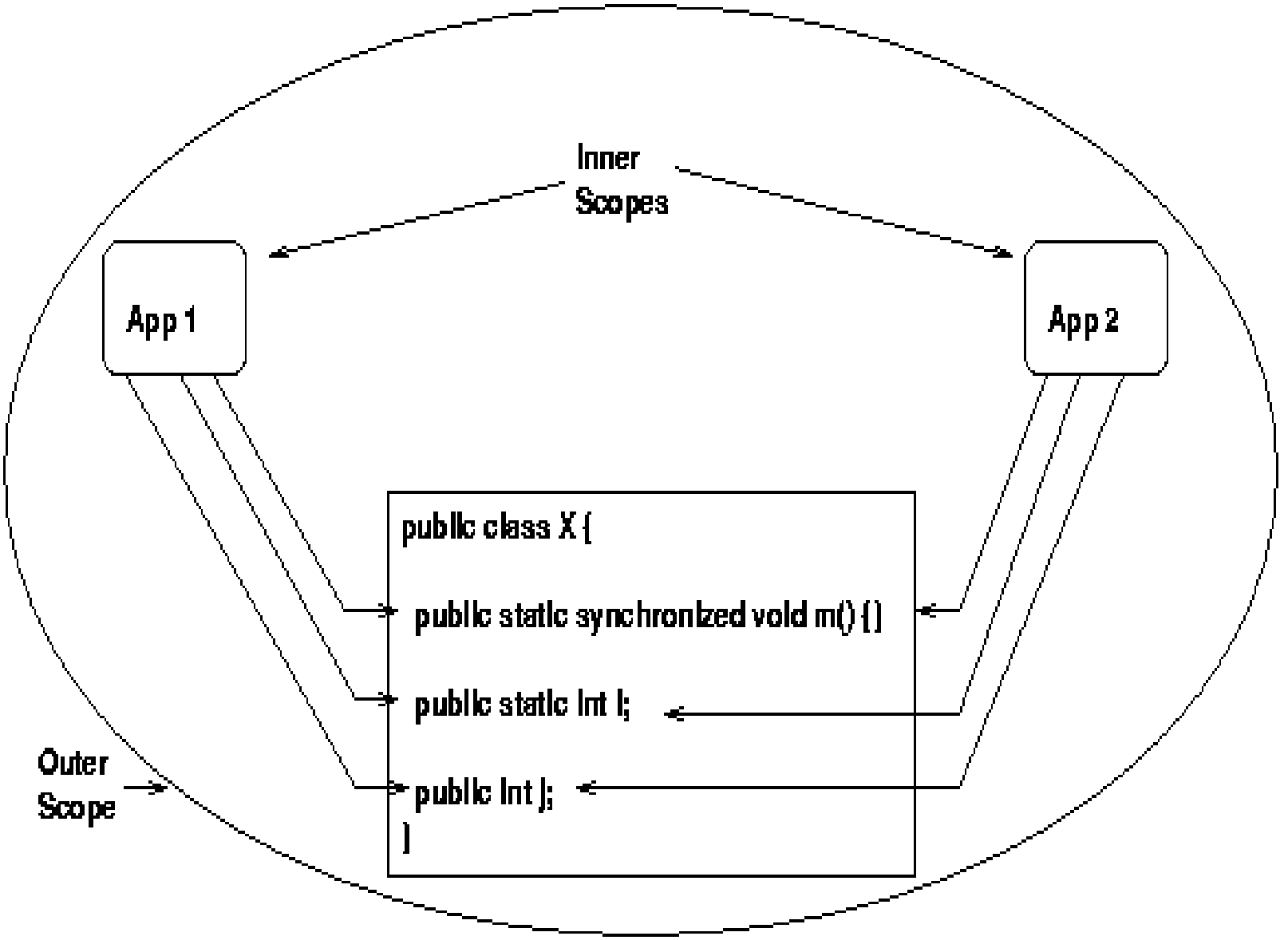
Sept 17, 2004

Overview

Isolate *noun*. pronounciation: ***isolet***.
1. A thing that has been isolated, as by geographic, ecologic or social barriers
   - *American Heritage Dictionary*

- Outline
  - Motivation
  - The Big Picture
  - API Details
  - Using the API
  - Patterns of Use
  - Wrapup

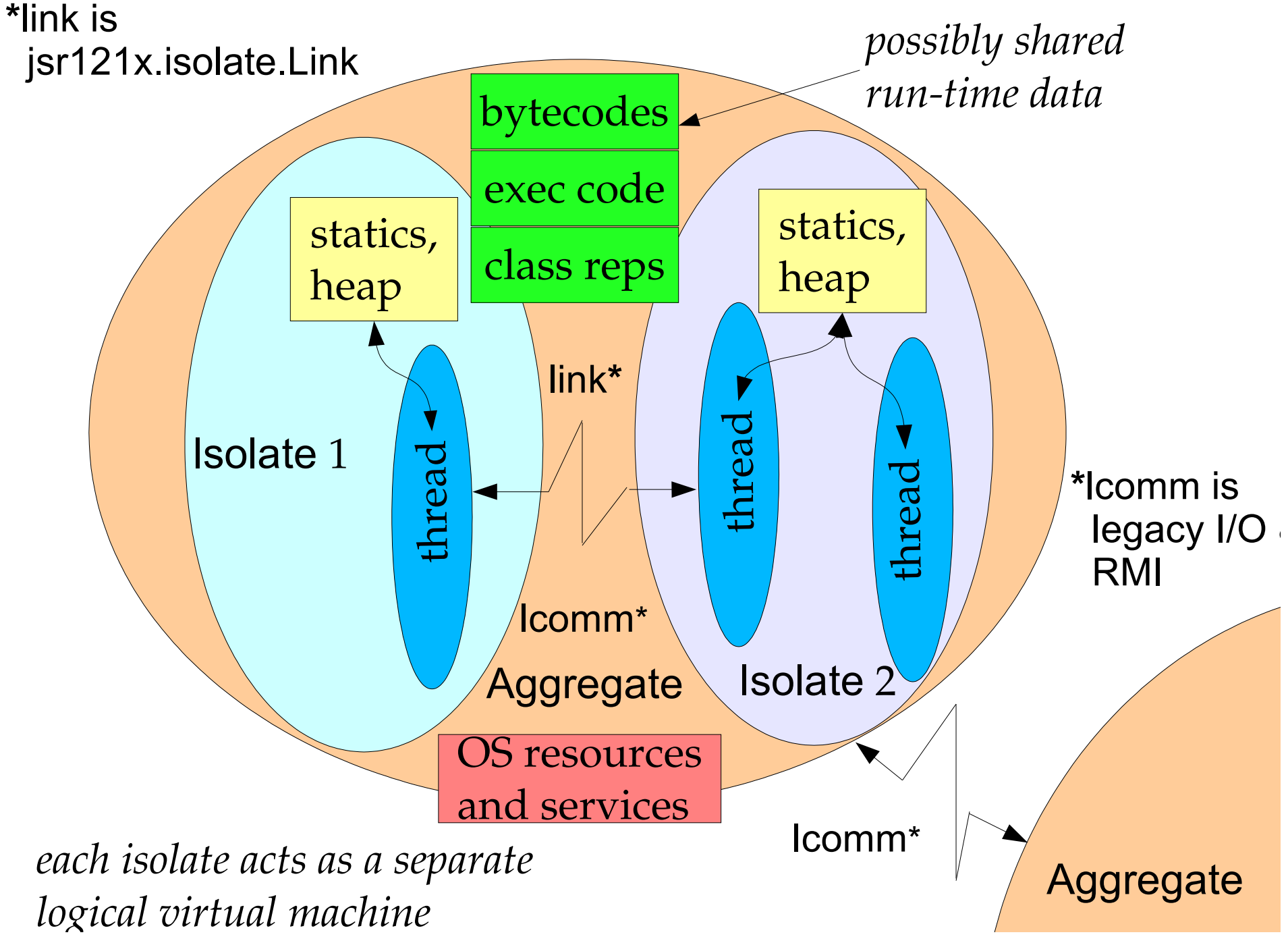
Problem: Undesired sharing



What's wrong with Sharing?

- Today, Servlets, Applets, etc run in same JVM
- Java Security helps prevent malicious interactions
  - But does not prevent all interference
- Unwanted sharing of static fields, and all objects reachable from them
  - Including many hidden inside JDK libraries
    - AWT, ThreadGroups, etc
- Unwanted dependence on global configuration
  - Resource settings, security policies, etc

Aggregates vs Isolates vs Threads



Benefits

- Management
  - Configure, monitor and reliably kill activities without disrupting others
    - Especially in container frameworks
  - Stay within Java; not out & in (Runtime.exec alternative)
- Security
  - Prevent interference via shared resources or communication
  - Simplify construction of obviously secure systems
- Scalability with less compromise

JVMs vs Isolate Aggregates

- Not necessarily a "single program" anymore
  - Each Isolate is a logical virtual machine
- A JVM is:
  - A running instance of the JRE
  - Strongly associated with a single OS process
- An Aggregate is:
  - A container of Isolates
    - An administrative and management boundary
  - A set of services and service guarantees
    - Bytecode execution and run-time functions
- Aggregate as a less ambiguous term

Isolating State

- Visible Per-Aggregate vs per-Isolate state
  - case-by-case analysis of statics, startup settings, global JVM state
    - See also Czajkowski et al MVM papers (in bibliography)
  - Spec requires very few global settings
    - All immutable: User identity, command-line settings
- Reliable lifecycle management enabled
- Native methods
  - JSR-121 does not strictly guarantee that bad JNI code will not crash some or all Isolates
    - Implementations can provide stronger guarantees, but at likely cost of crossing address spaces for JNI calls

Security

- Per-Isolate Security Managers
  - Can arrange different managers and policies for different Isolates
  - Common default security policy files
- Checks for creating, controlling and communicating between isolates
  - IsolatePermission controls access (CDC&J2SE)
- Aggregate runs under single User identity
  - No Unix-style substitute-user capability
- Capability style inter-isolate communication
  - Must create a Link to communicate, and must posess Isolate handles to create Link

Implementation Styles

- One Isolate per process: 1:1 Style
  - Sharing of runtime data possible (e.g. class data sharing)
- All Isolates in one process: N:1 Style
  - Isolates still get own versions of all statics/globals
    - including AWT thread, shutdown hooks, ...
- N Isolates scheduled onto M processes: N:M Style
  - At any one time only M isolates doing work
- LAN Cluster Aggregate: Research Topic
  - Isolates on different machines: ***one admin domain***

All Implementation styles compatible with the spec!

API Structure

- Base Package jsr121x.isolate
  - Interface
    - Sendable
  - Classes
    - Isolate
    - IsolateParameters
    - Link
    - DataMessage
    - StatusMessage
    - StringMessage
    - CompositeMessage
  - A few exception classes
- Additional Support
  - NOT in small J2ME
- jsr121x.isolate.tbd
  - IsolatePermission
  - ObjectMessage
- jsr121x.isolate.{io,nio}
  - IOParameters
  - Classes for wrapping IO handles etc
- jsr121x.isolate.util
  - Support many links without thread per link

Primary Classes

- **`public final class Isolate implements Sendable`**
  - Create easily with just name of class with a "**`main`**" and String args or with additional context
  - Methods to start, terminate and query isolate
  - Created isolate can get its parms and starting links
- **`public class Link implements Sendable`**
  - Create pipe-like connections between source and destination Isolate endpoints
  - Rich set of message types from byte arrays to SocketChanels, plus:
    - Isolate and Link instances
    - Composite messages of arbitrary complexity sent atomicly

Starting Isolates

- Isolate creation establishes existence
  - Isolates may (but need not) perform resource allocation and internal initialization upon creation
- Static initializers, then main run at **`start`**
  - Isolates may continue initialization before running
  - All classes are loaded in new Isolate's context
- Failures detected before running user code result in exceptions at creation or start time
  - Cannot be sure whether the same exceptions will be thrown at the same points in all Implementations
- Other failures merely terminate the Isolate

Easy Isolate Use

```
void runProgram(String classname,
                String[] args) {
  try {
    new Isolate(classname, args).start();
  }catch (IsolateStartException ise) {
   // ...
  }
}
```

Configuration

- Inheriting execution contexts
  - Easy way: Just String[] arguments
  - Simple name/value pairs can be included
    - Some map to Properties
  - Standard stream bindings fill out context
- Other Mechanisms OK to use too
  - Contained Isolates may obtain additional configuration parameters via JNDI or other means
  - Frameworks can supply a common main that establishes context and then loads application

Stopping Isolates

- Preserves distinction between exit and halt
  - **`exit`** causes Isolate to run shutdown hooks etc
    - Does NOT guarantee eventual termination
  - **`halt`** causes sure, abrupt termination
  - Isolates may also terminate for the usual reasons
  - Aggregate shuts down when ALL Isolates do
- Monitoring lifecycles
  - Receiving start, exit, terminated events
- Not hierarchical
  - Parents may terminate independently of children
    - Can layer on methods to await termination

Simple Link Use; Custom Policy

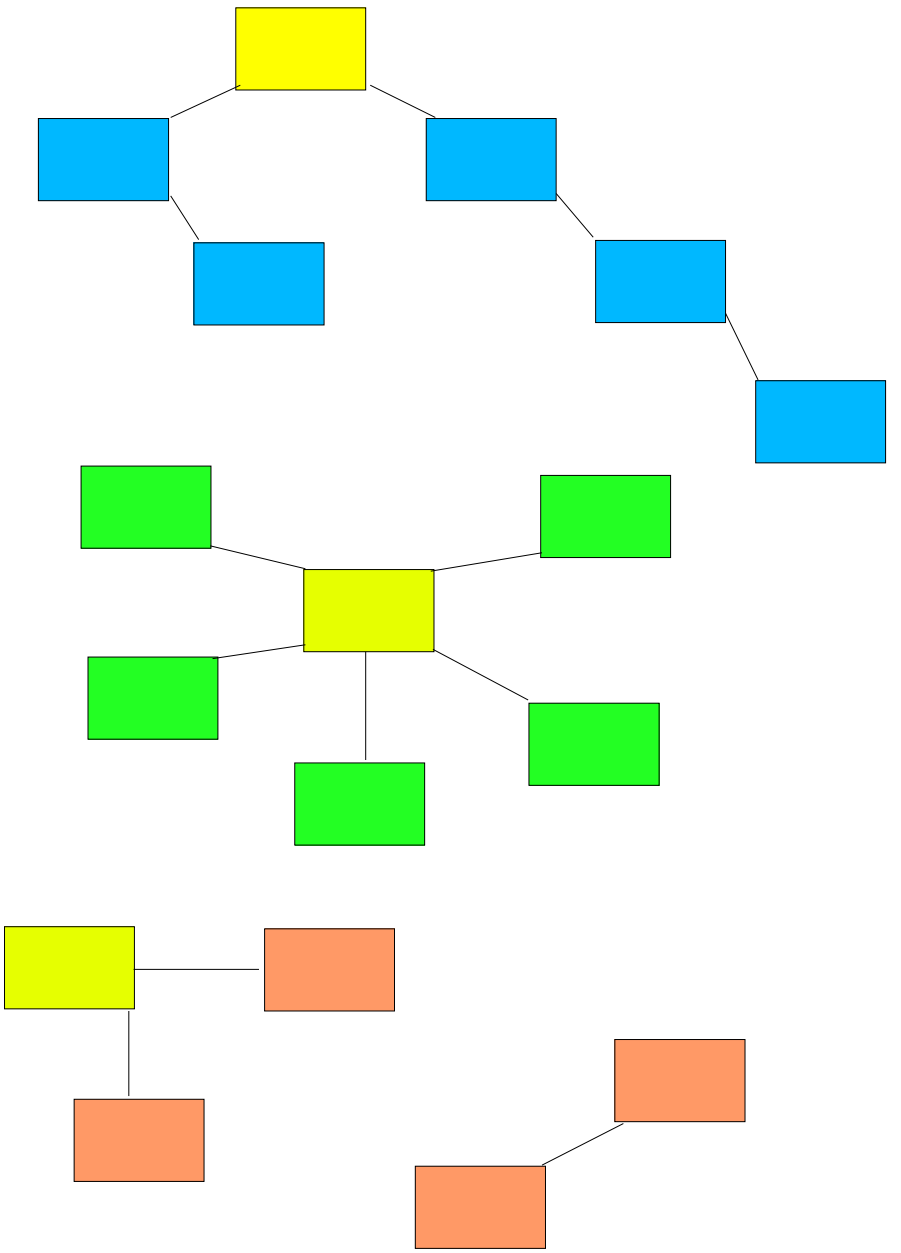

```
Class Runner {
  Link data;
  Isolate child;

  CompositeMessage getMessage() {
    return data.receive();
  }

  StatusMessage doit(String mCls, String[] mArgs,
                             String[] secMgr) {
    IsolateParameters context = new
                                IsolateParameters(mCls, mArgs);
    context.setContext(
       “jsr121x.properties.java.security.manager”, secMgr);
    child = new Isolate(context);
    data = Link.newLink(child, Isolate.currentIsolate());
    StatusLink s = child.newStatusLink();
    child.start(new Link[] { data } );
    return s.receive();
  }
}
```

Flexible Topology

- App frameworks can impose policies:
  - Hierarchical
  - Centralized
  - Ad-hoc
  - Can add monitoring for application-specific events and/or tie to external monitoring



Usage Patterns

- Partitioning applications
  - Contained applications
    - Groups of Applets, Servlets, Xlets, Midlets, etc can run inside Isolates
    - Container utility services can run inside Isolates
  - Service Handler Forks
    - ServerSocket.accept can launch handler for new client as Isolate
    - Pools of "warm" Isolates

More Usage Patterns

- Fault-tolerance
  - Fault detection and re-activation frameworks
  - Redundancy via multiple Isolates
- CSP style programming
  - Always use Isolates instead of Threads
  - Practical only for coarse-grained designs
- Parallel execution on cluster JVMs
  - Java analogs of Beowulf clusters
    - Can use MPI over Links
  - Need partitioning and load-balancing frameworks

Milestones

- Assembled a strong and involved expert group
- Java Community Review Draft
  - Partial N:1 style implementation by Sun Research
- Public Review Draft
  - 1:1 style implementation written by Miles Sabin through special arrangement with Sun
  - N:1 style partial implementation written by Pat Tullmann and the U. of Utah Flux group
- APIs redesigned (mostly just refactored)
  - Package boundaries clean
  - Simplifications, fixes and improvements (ongoing)

Credits

- Sun Task API group
  - Greg Czajkowski
  - Bill Foote
  - Hideya Kawahara
  - Tim Lindholm
  - Glenn Skinner
  - Pete Soper
- Past JSR-121 EG Members
  - Beth Hutchison, IBM
  - Peter Donald, Apache
  - Jens Jensen, Ericsson
  - Pat Tullman, U of Utah
  - Kumanan Yogaratnam, Espial
- Current EG Members
  - Dat Doan, Espial
- (Current EG cont'd)
  - Richard Houldsworth, Philips
  - Norbert Kuck, SAP
  - Doug Lea, SUNY Oswego
  - Michey Mehta, HP
  - David Raymer, Motorola
  - Miles Sabin
  - Pete Soper, Sun (lead)
  - David Unietis, Oracle
  - Matthew Webster, IBM

Resource Management

- Not specified in JSR-121
  - ***NO*** guarantees about scheduling, heap mgt, etc
  - Hints are possible via IsolateParameters
- Current Sun Research
  - Sun technical report TR-2003-124 (in bib.)
    - Designed with joint effort by reps from Research, Solaris Software, Java Software and Ciaran Bryce of U. of Geneva
  - More papers in bibliograpy and more coming