Designing Scalable High Performance Rich Clients from the Trenches

Rob Ratcliff

Partner/Consultant/Lead Engineer
FutureTek Net Services LLC http://www.futuretek.com/

TS-3723

Session Goals

Design Lessons from the Trenches

Learn practical design lessons from the development of a challenging networked rich client application

About FutureTek LLC

- Founded in 1995
- Projects and Technologies
  - High performance rich networked Swing clients
  - Distributed Multi-Disciplinary Optimization Framework for Boeing, MDOPT
  - CORBA integration
  - Database applications
  - Java based web projects
    - On-line ordering, radio show scheduling, call center, inventory management
  - Multi-player trivia game applet
- Co-founded Austin Java Users Group
  http://www.austinjug.org

Agenda

- Overview of the “NEWS” Project
- Architecture Overview
- Comparison and Contrast of Design Solutions
- War Stories and Lessons Learned
- Summary

NEWS Project Background

- Design GUI to control a network of sensors and display the resulting engineering data in near real time
- Overall Design Approach
  - Mockup screens using NetBeans to allow customer to interact using mock data
  - Java Swing Toolkit for GUI (rather than AWT)
  - Stateful high-performance binary protocol and asynchronous messaging
  - Simple command+name/value pair control language
  - Socket per session mimicking a distributed service factory (made demultiplexing of events easier too)
  - Develop mock server to accelerate development

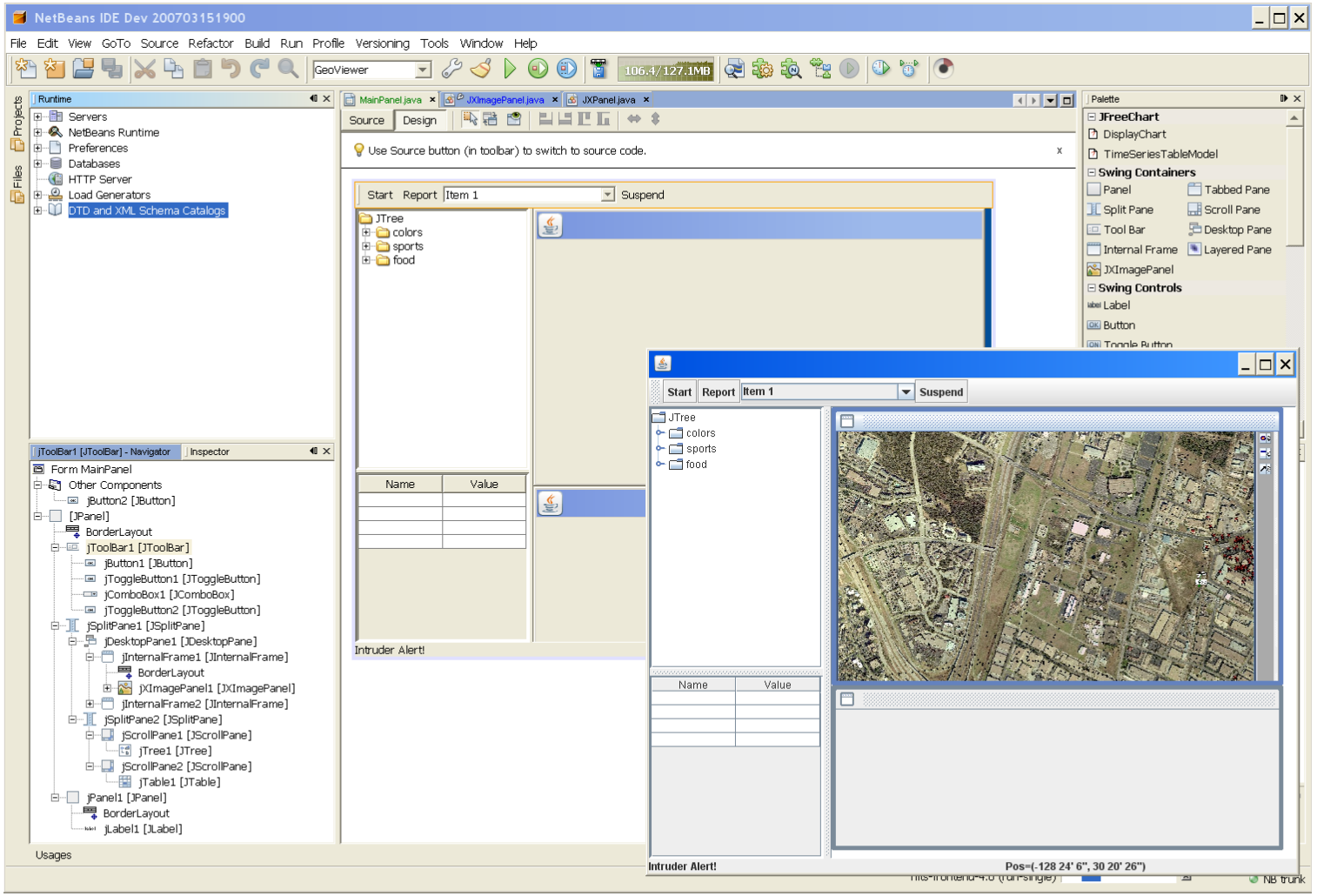
Application Display Requirements

- Support streaming data (positions and speed, signal, status, audio etc.)
- Display time-varying positions and other sensor-related data on a world map
- Display data in time order
- Synchronize all views in time
- Synchronize selection in all views
- Minimal mouse clicks to access functions
- Display all relevant data up front

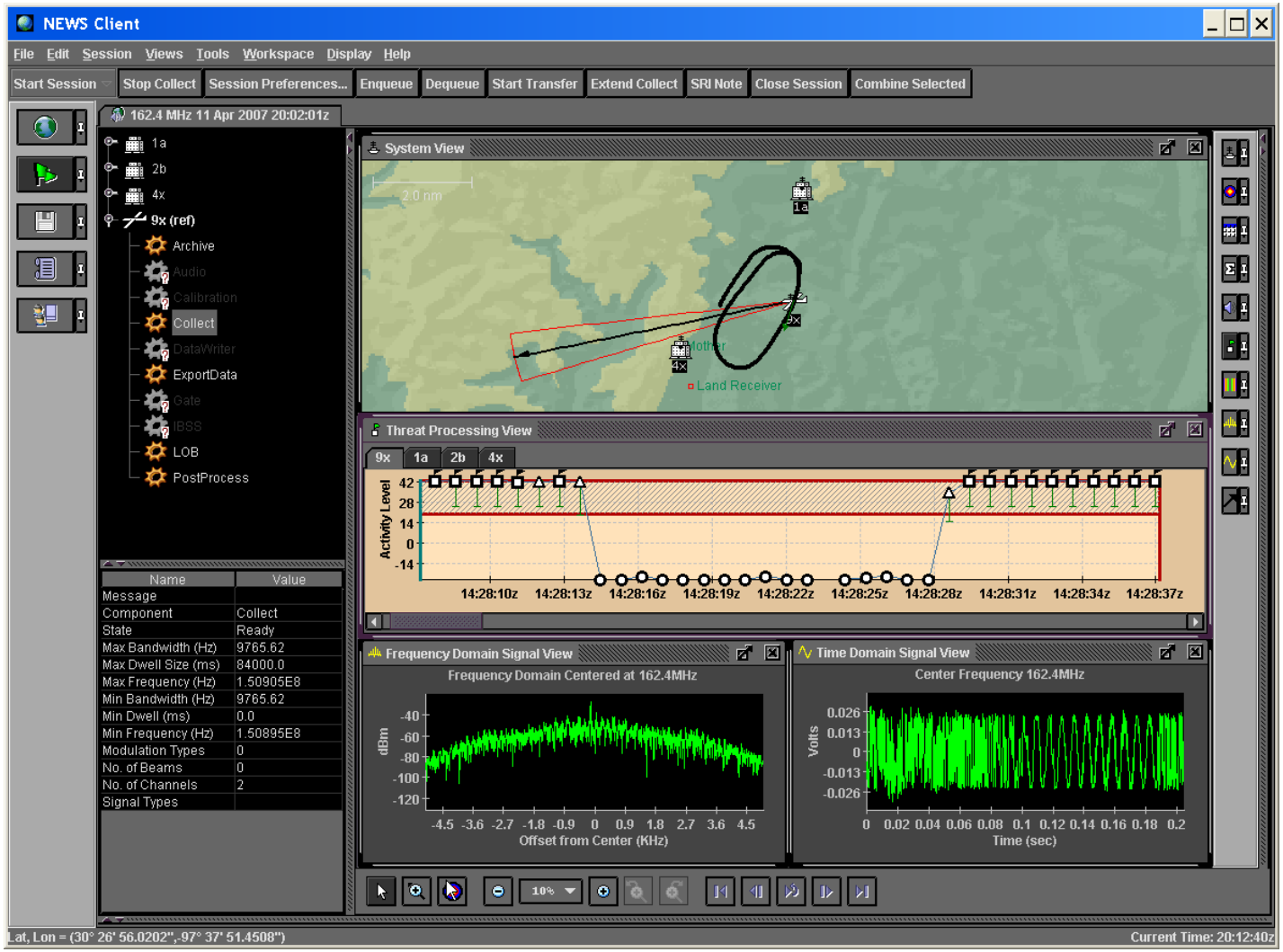
Design Scalability Defined

- **Feature**
  - GUI Layout
    - Toolbar space
    - Menu Structure
    - Screen real estate
    - Feature Navigation
  - Hooking in new components
  - Leveraging what others have done
- **Duration** (long execution runs and multiple sessions)
  - Memory Performance
  - CPU Performance
  - GUI Responsiveness
- **Development**
  - Practice type safety first (avoid string based paradigms)
    - Ease editing through IDE auto-completion
    - Ease refactoring
    - Ease debugging through compile-time error checking
  - GUI-builder and XML driven screens for easier long term maintenance
- **Deployment**
  - Easy configuration as number of settings increase

NEWS Initial Mockup Rapidly Generated Using NetBeans GUI Builder



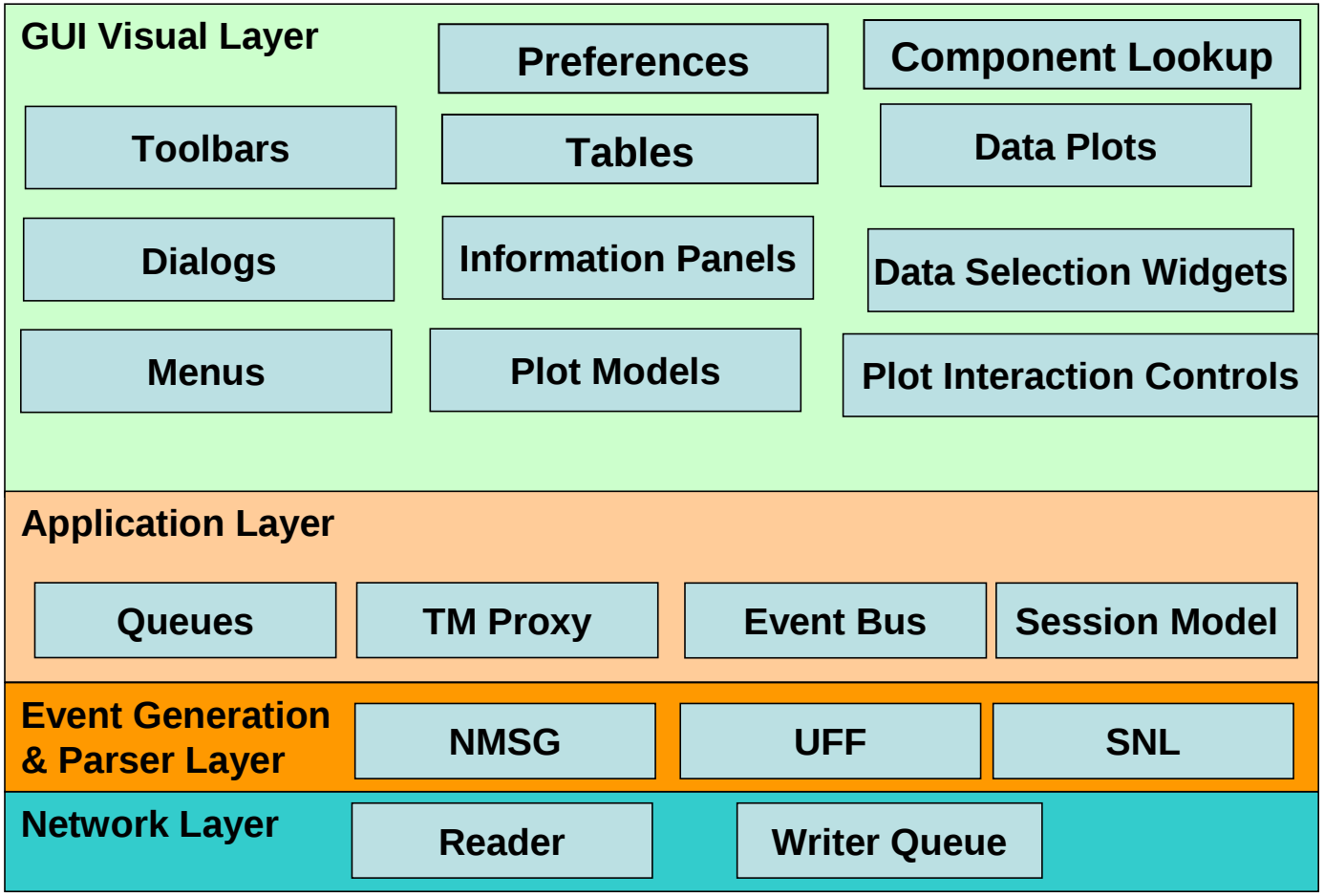
NEWS Networked Sensor Application



DEMO

NEWS Intruder Detection Scenario

Architecture Forensics



I’m an Enterprise Unto Myself

Large GUI applications share many of the problems that distributed component architectures have

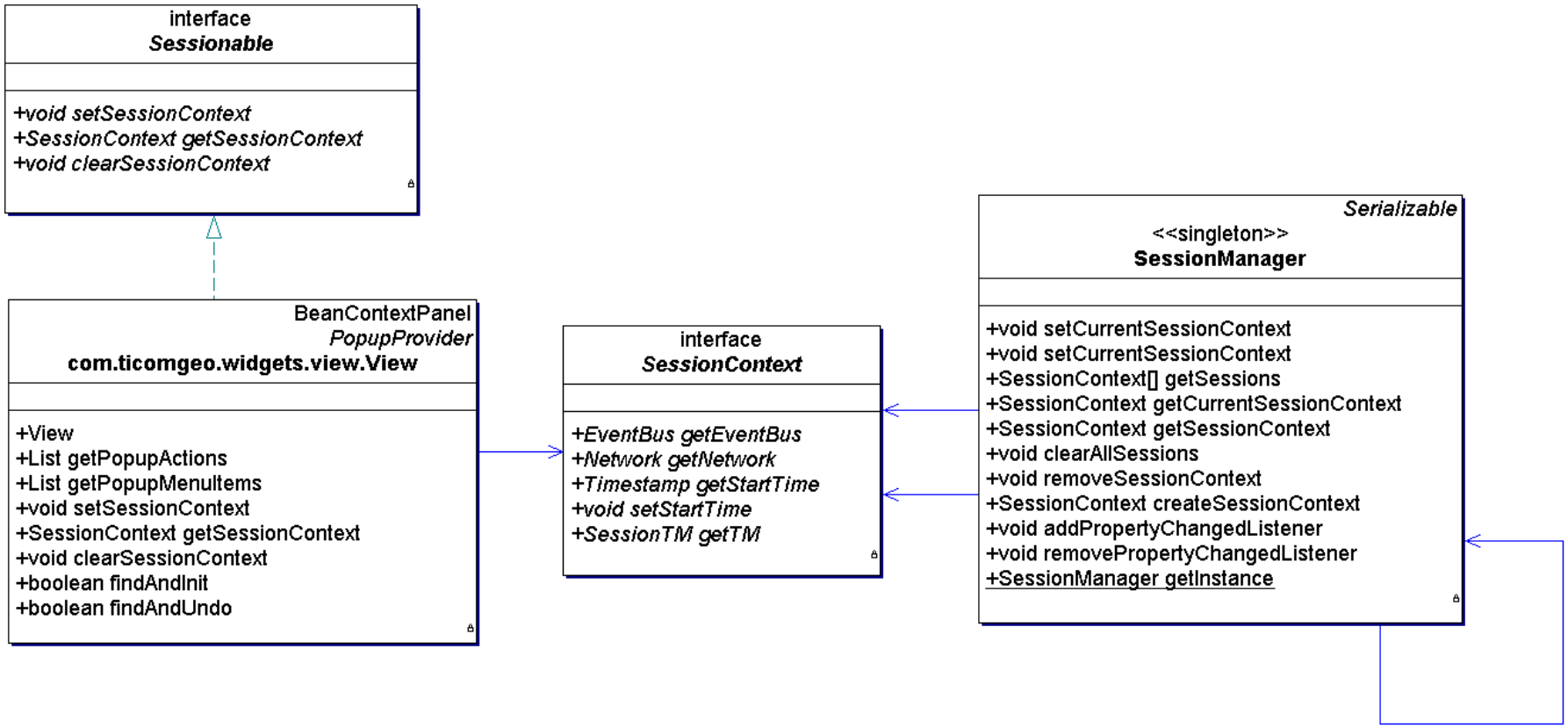
- Complex threading issues
- Lookup and hookup with “distant” components and models
- Asynchronous notification
- Transactions
- Persistence
- Data mappings and graph navigation

Where in the World is My Component?

- Dependency Injection (Constructor or Setter)
- Singleton
- BeanContext
- Lookup Services
- Event Bus

Dependency Injection before Injection was Cool

SessionContext idea borrowed from the earlier EJB specification

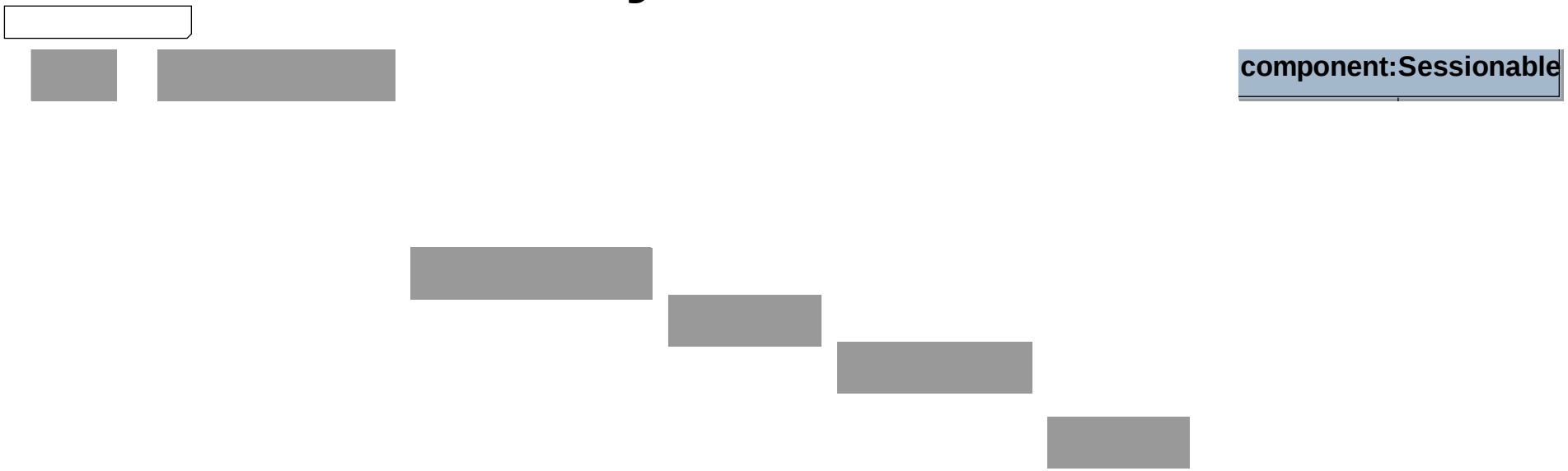


SessionContext

Life Cycle Management

- Creation
  - Create Queues and Worker Threads
  - Data Models
  - Register data models with event bus
  - Component listener hookup
- Destruction
  - Thread shutdown
  - Socket disconnect
  - Component listener removal (to avoid memory leaks)

Session Life Cycle



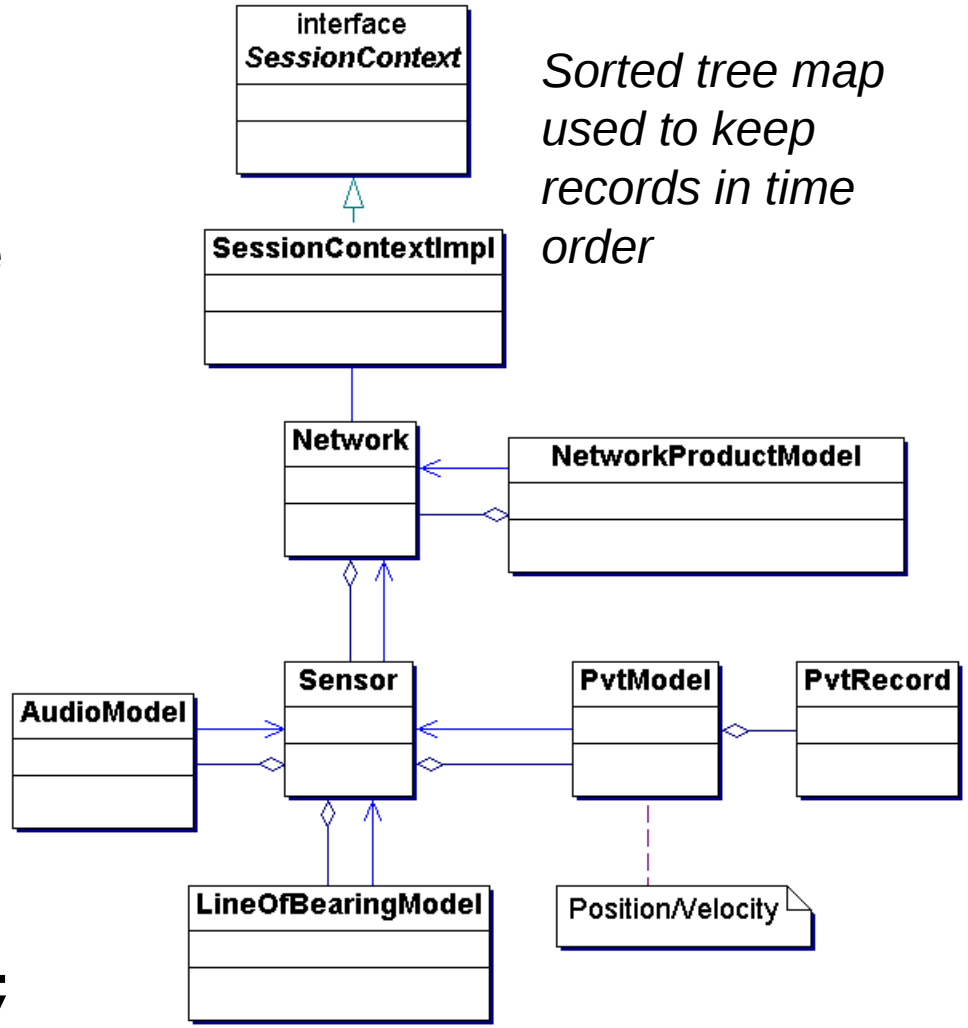
Domain Specific Data Model

Advantages

- **Easy to navigate**
- **Domain specific data rules and state**

Disadvantages

- **More coupling due to dependencies**
- **Harder to test**



Sorted tree map used to keep records in time order

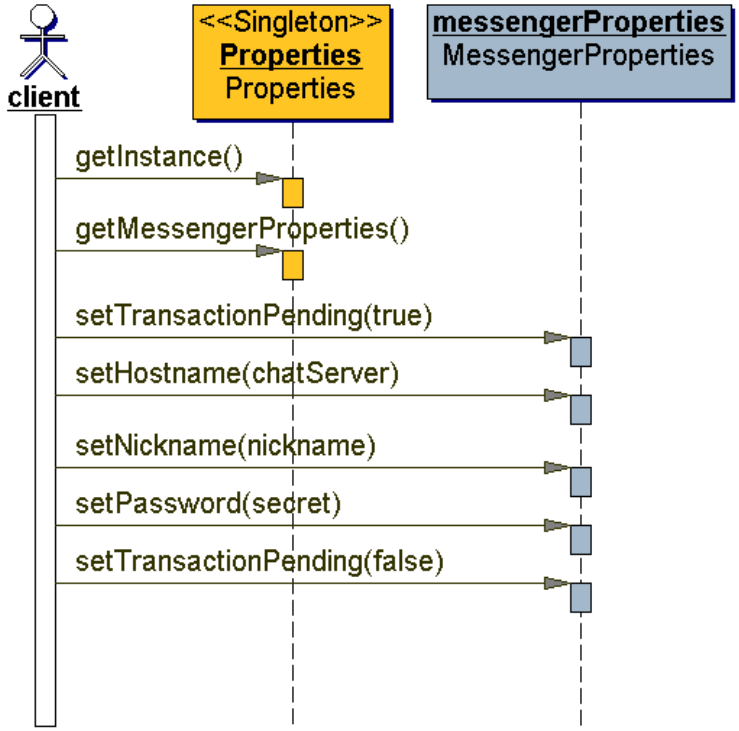
```
Usage:
sessionContext.getNetwork()
.getLocalSensor()
.getPvtModel().getRecords();
```

Singleton Example

Type-Safe Preferences Management

- Goal: Make preferences easily accessible to programmer without having to know magic strings
- Type safe façade class to access properties for a given module
- Use one property file

Note: Java Beans BeanInfo framework very useful for dynamically setting and serializing property values and auto-generating property sheets



Singleton Tradeoffs

Advantages

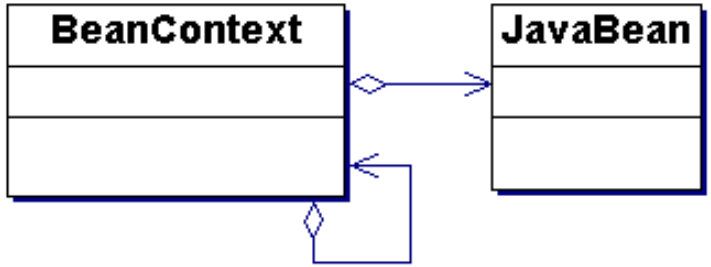
- Simple concept
- Globally accessible
- Ensures one instance

Disadvantages

- Leads to tighter coupling
- Ensures one instance

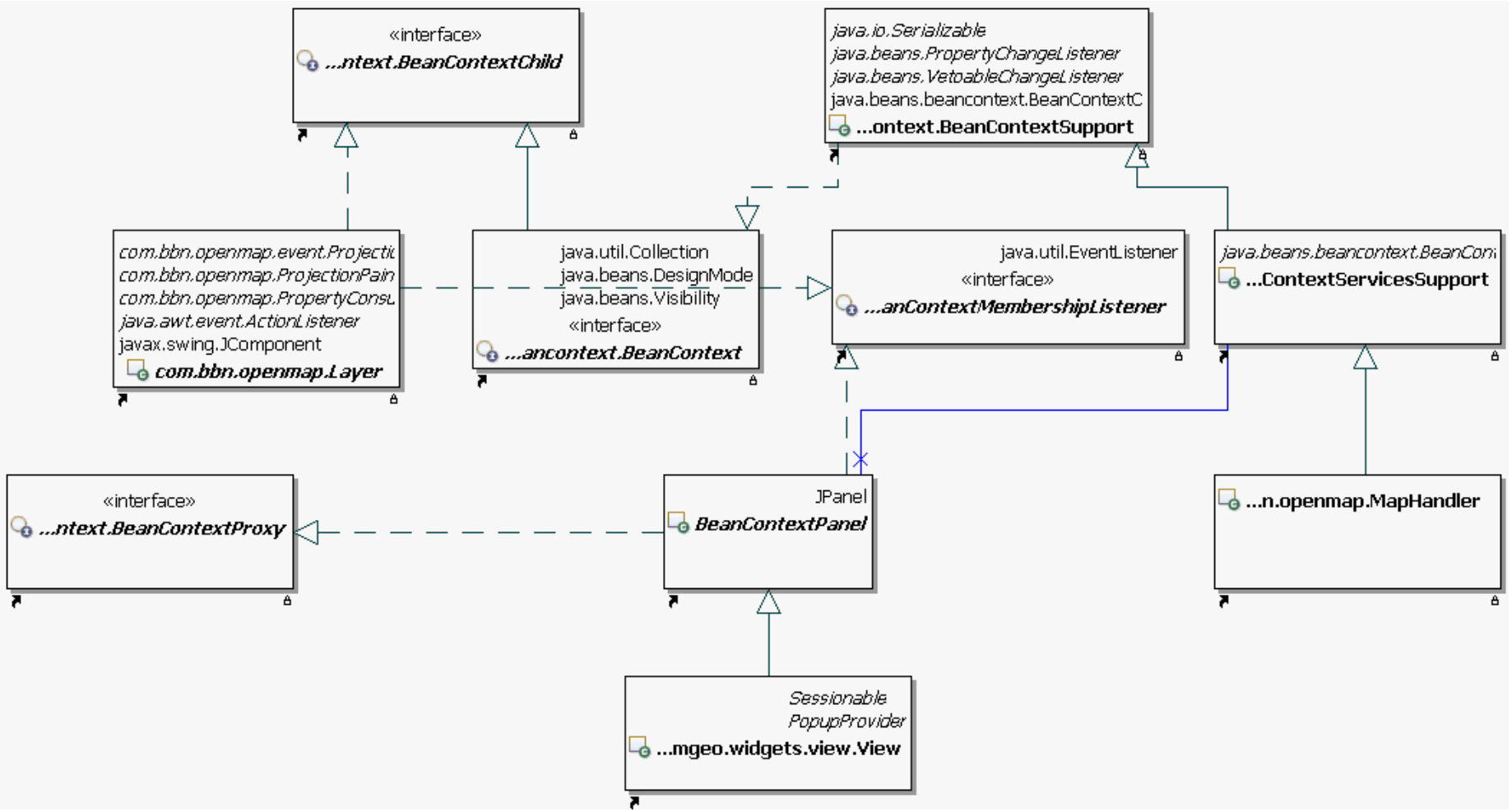
BeanContext Architecture and Navigation

- Based on composite design pattern



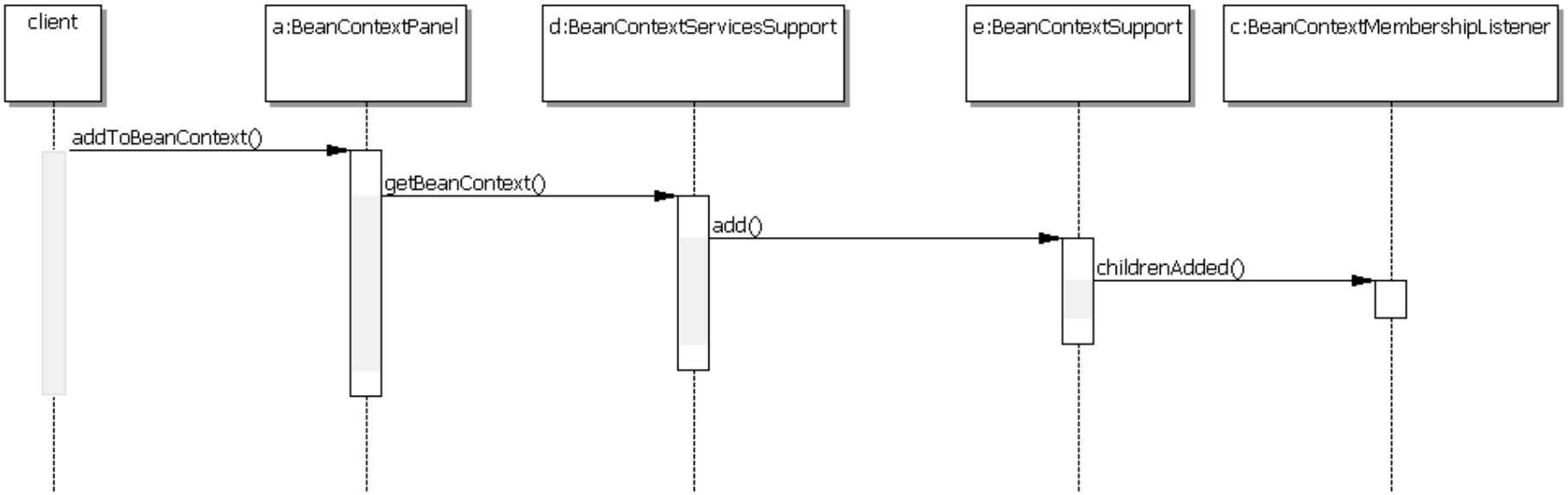
Note: OpenMap framework built on java.beans.BeanContext infrastructure

Sample BeanContext Interfaces and Support Classes and Consumers



Adding a Bean to the BeanContext

Changed



Note: The OpenMap framework delegates all added context change notifications to the findAndInit(Obect o) to reduce the number of user-implemented methods

BeanContext Tradeoffs

Advantages

- Flexible composite structure
- Part of the standard JDK distribution
- Allowed easy integration with the OpenMap framework

Disadvantages

- Rather large set of interfaces and methods to implement
- A bit complicated
- Custom code required to support hierarchical notifications and searches

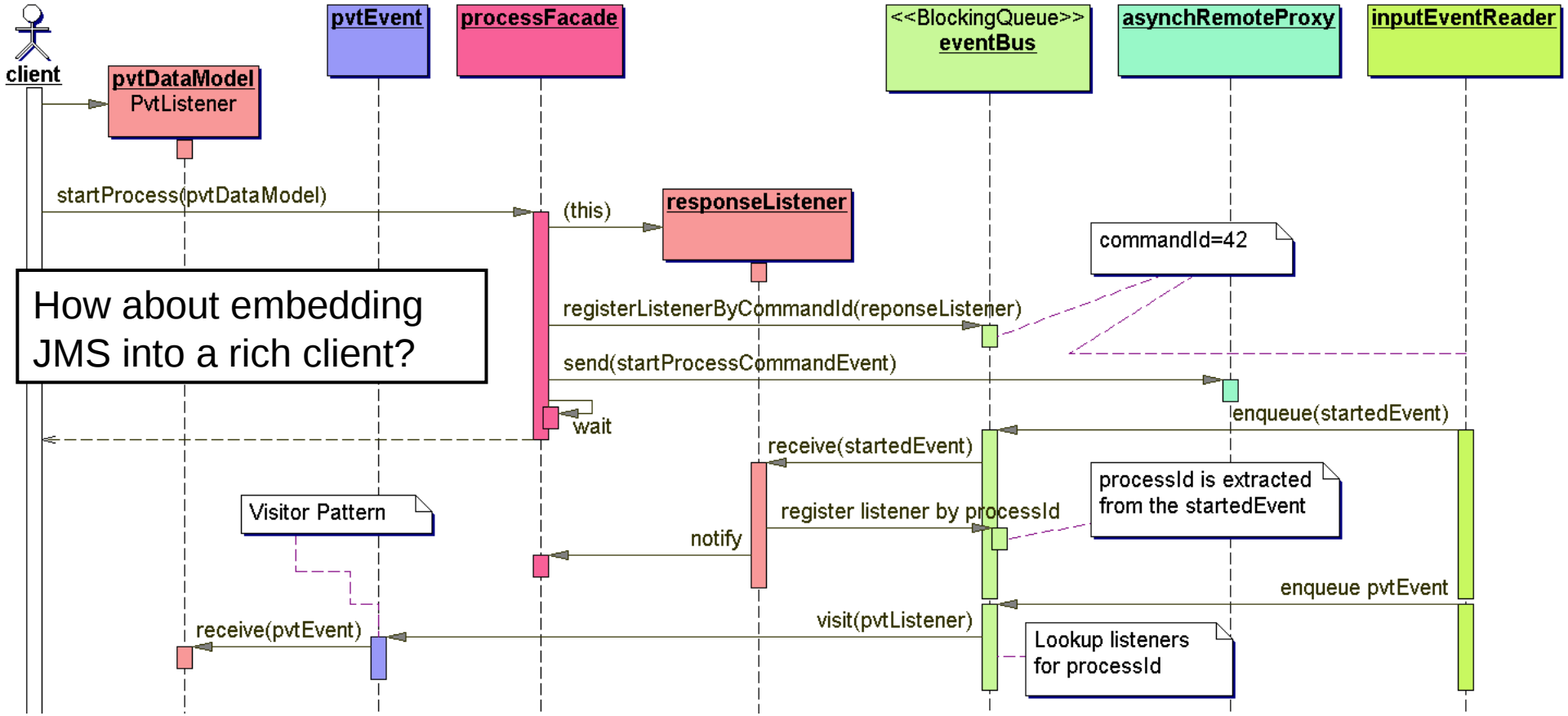
Example Lookup Services

- NetBeans style Lookup
  - `(ChatProxy)Lookup.lookup(ChatProxy.class);`
  - `(ChatProxy)Lookup.lookup(`
`new Template(ChatProxy.class, “mychat”));`
- JNDI Lookup
  - `(ChatProxy)context.lookup(objectName);`
  - `(ChatProxy)directory.search(contextName,`
`attributes)`

Keep Asynchronous “Remote Calls” Simple Through Synchronicity

- Listener and correlation ID pattern to block for response when protocol is asynchronous or message based
- Foxtrot keeps GUI lively (Concurrent worker vs. Single worker)
- Deliver events in correct thread

Synchronous Pub/Sub Paradigm with Asynchronous Communications



InvokeLater Insanity

"*Doing the same thing over and over and expecting different results" – Ben Franklin*

- Expecting developers to consistently use SwingUtilities.invokeLater() is wishful thinking at best
- Framework should take care of this automatically

Inter-component Communications using Event Bus

- Listeners implementing SwingListener interface receive delivered events on EDT

- Listeners register for Swing events by type
- Events pushed onto bus

Using Event Bus for Inter-Component Communication Tradeoffs

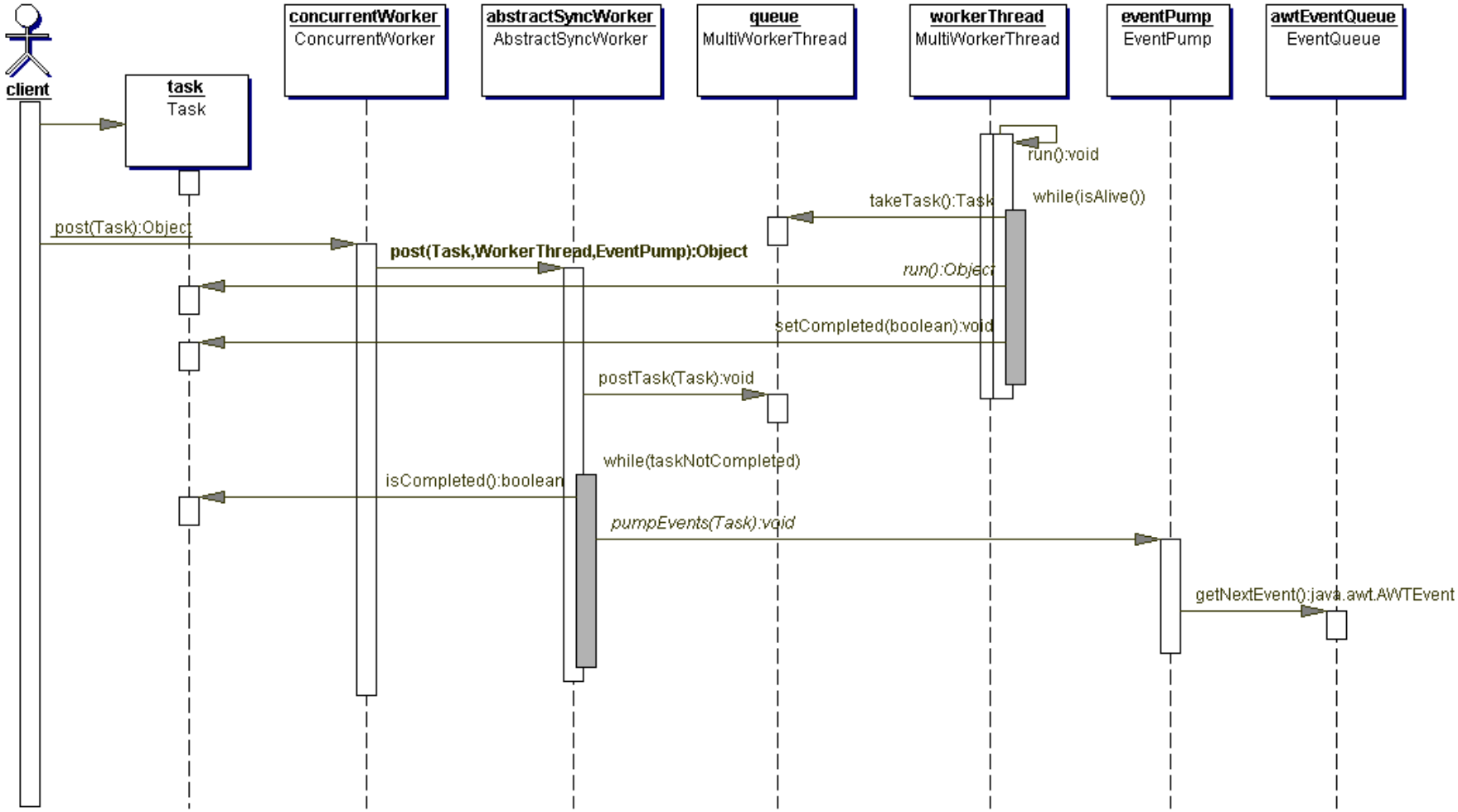
Advantages

- Supports multiple producers of a given event type
- No knowledge of producer required
- Can hook up directly with producer via “source” of event

Disadvantages

- Proliferation of event types
- Slower performance than direct method calls
- Demultiplex desired producer

How does Foxtrot Work?



Embedding Foxtrot into Remote Interface Proxy (Stub) using Dynamic Proxy

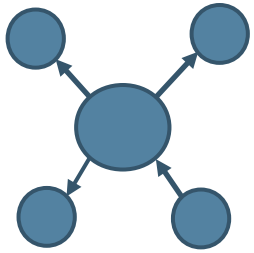
```
ChatProxyHandler chatProxyHandler =
   new ChatProxyHandler(chatProxyDelegate);


ChatCommand proxy = (ChatCommand)
   Proxy.newProxyInstance(loader,
      new Class[] { ChatCommand.class },
            chatProxyHandler);
```

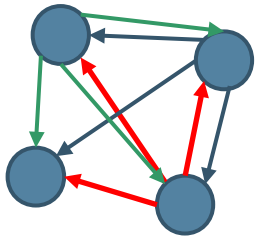
Dynamic Proxy Continued

```
class ChatProxyHandler implements InvocationHandler {
  public Object invoke(Object proxy,
    final Method method, final Object[] args) {
     this.fireBusy(true); Object value = null;
     try {
      if (!SwingUtilities.isEventDispatchThread()) {
       value = method.invoke(delegate, args);
      } else {
       value = ConcurrentWorker.post( new Task() {
         public Object run() throws Exception {
            return method.invoke(delegate, args);
         }
       });
      }
     } finally {
       this.fireBusy(false);
```

Global Selection Notification

- Desired Behavior:
  - In a time-based simulation, all views need to be time synchronized to avoid confusion
  - Selected points in one view may need to cause selection of points in other views
- Causes
  - Current Timestamp changes
  - User selects data that may or may not cause a timestamp change
- Solutions
  - Spoke and hub
  - Point-to-point





Low Maintenance GUI Forms

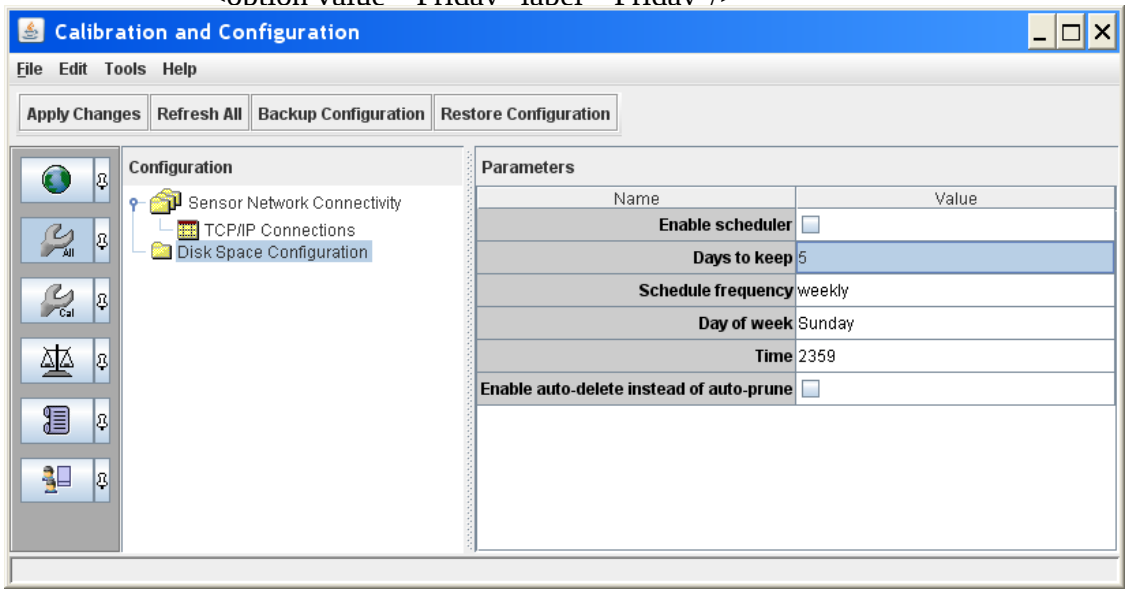
Higher Team Development Scalability

- GUI builder generated forms like NetBeans' Matisse
- Forms generated from XML data descriptions
- Forms generated from Java Beans and BeanInfo

Generating Forms Automatically with XML

```
<config>
    <boolean name="enabled">
      <value>false</value>
    </boolean>
    <long name="expiration">
      <value>5</value>
    </long>
    <string name="when_to_run">
      <value>weekly</value>
    </string>
    <string name="day_of_week">
      <value>Sunday</value>
    </string>
    <string name="time_to_run">
      <value>2359</value>
    </string>
    <boolean name="delete_file">
      <value>true</value>
    </boolean>
</config>
```

```
<configDescriptor>
    <stringDescriptor name="day_of_week" privilege="operator" label="Day of
week" recommended="Sunday">
      <description>If weekly, sets days of week</description>
      <options>
            <option value="Sunday" label="Sunday"/>
            <option value="Monday" label="Monday"/>
            <option value="Tuesday" label="Tuesday"/>
            <option value="Wednesday" label="Wednesday"/>
            <option value="Thursday" label="Thursday"/>
            <option value="Friday" label="Friday"/>
```



War Stories and Lessons Learned

- GUI locked up!
- Start session dialog comes up blank!
- System won't respond!
- GUI is hogging the CPU!
- GUI is hogging memory!
- GUI is hogging file descriptors!
- Sessions won't start due to a null pointer exception!
- I'm seeing thousands of index out of bounds exceptions!

Post Mortem Analysis of Bug History

- 1600 total bugs/enhancements over 4 years
- 323 P1 categorized bugs
- 67 GUI freezes/deadlock bugs
- 82 Null pointer exception bugs
- 68 Index out of bounds

Embarrassing Lockups

- Primary Reasons
  - Developer forgot to use InvokeLater leading to deadlock
  - Synchronous calls to non-responsive server made in EDT
- Remedies
  - Check wrong thread with custom repaint manager
  - Foxtrot – Concurrent or Single Worker model
  - SwingWorker – Asynchronous solutions
  - Don’t burden programmers, embed this in framework
    - Dynamic Proxy and Chat application
    - “SwingListeners” – Event distributor automatically deliver events to listener on EDT
    - Practice DRY (Don’t Repeat Yourself)

Teach Testers about “kill –QUIT”

- Deadlocks are hard to find, especially when they are intermittent
- Compile code with debugging symbols turned on
- Tell QA folks to use “kill –QUIT” during a lockup to produce a detailed stack trace

Beating Death By a Thousand Paint Strokes

"There are simply too many notes, that's all. Just cut a few and it will be perfect!" – Emperor Joseph II to Mozart in Amadeus

- Scheduled repaints
- Eliminate off screen label/icon/line paints
- Paint only "dirty" regions and only when "dirty"
- Use off-screen buffered images for static map layers like the coastlines of the world (See OpenMap's BufferedImageRenderPolicy for an example)
- Use polyline and buffered images to avoid loading up X-server
- Watch out for slowdowns caused by semi-transparent colors when displayed over remote X
- Change Java2D command line parameters for optimal remote X operation, such as:

```
-Dsun.java2d.pmoffscreen=true|false
```

Populate Data Models Before Hooking up Views

- Post-processing of large data files can cause thousands of events to fire in seconds
- Hook up GUI components after data is delivered to all models
  - Greatly speeds up responsiveness of GUI
  - Allows user to monitor and cancel the request

Debugging Paint Slowdowns

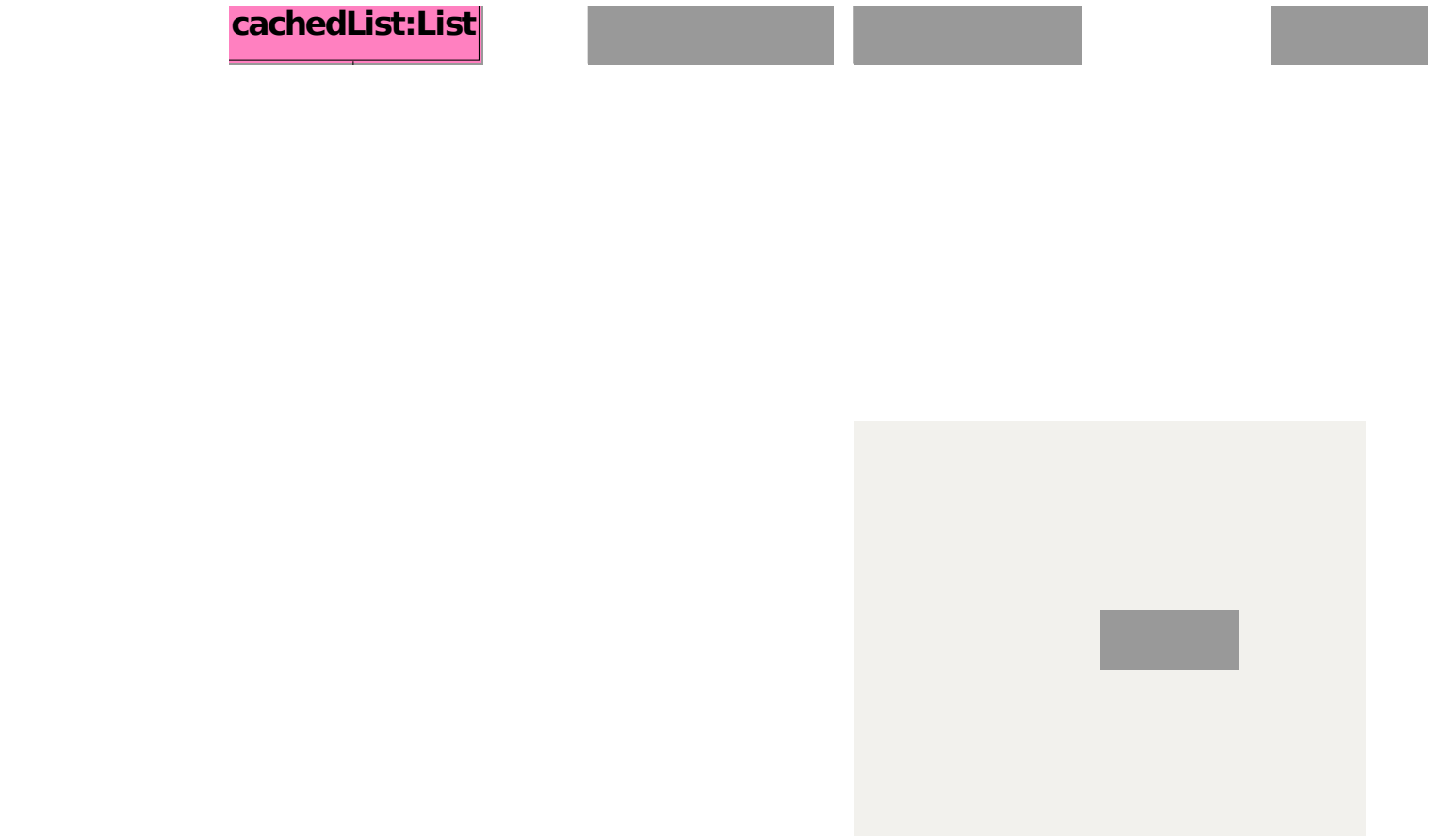
Top is your friend!

- Iconify each view or cover a given view with another window and watch CPU time
- Note how performance degrades over time
- Look for long lists of labels or icons even outside of graphics clip region
- Look for events causing multiple repaints per second
- Check whether slowdowns occur over remoted X connections

Strategies for Handling Large Data Generated by Long Sessions

- Cache older data to disk
  - Cache manager approach using evictor pattern
  - NetBeans approach for log files using NIO's memory mapped IO (See **`org\netbeans\core\output2`** or Tim Boudreau)
- Re-query server for data that has been purged

Example Evictor Cache Management

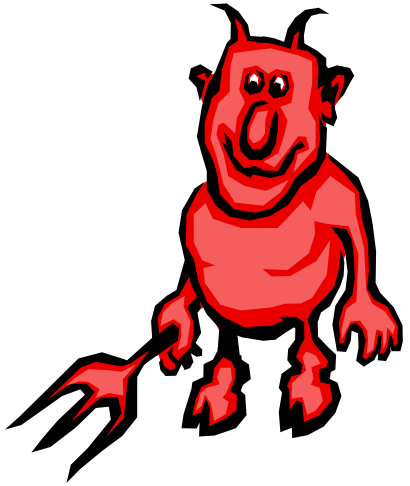


Use “JTable Style” Model/View for Long Duration Dynamic Time Series Plots

- Synchronizing large sets between plotting package and domain model can be problematic
  - Impedance mismatches
  - Error-prone
  - Complicated
  - Hard to get performance right
- Preferred approach now
  - JTable/TableModel style approach – Performs well and simple to track state
  - Query original model for state info for each view based on the time slice shown
  - Custom Renderers can be used to “rubber stamp” data based on state

Avoiding Null Pointer Hell

- Null Object Pattern
  - Zero Length Arrays or Collections
    ```
    return new ArrayList();
    return new String[0];
    ```
  - Do nothing object
    ```
    return BLANK_AUDIO_RECORD;
    return NoOpProxy;
    ```



- Throw an exception rather than returning null unless you need high performance to force programmatic checking
- Immutable Objects with a builder
- Assert for null on arguments (Use aspects or annotations to relieve coding tedium)

Addressing Slow Networks

- Multiple sockets allow interleaving of data and command requests/responses
- Handle socket disconnects and reconnects gracefully
- Gracefully handle timeouts (have timeouts!)
- Queue requests for data and return asynchronously
- Cache data to local disk rather than refetching from remote server

Rich Client Frameworks to Follow

- NetBeans Platform:
  http://www.netbeans.org
- JavaDesktop, Fuse, SwingX
  http://www.swinglabs.org
- Spring Rich Client
  http://spring-rich-c.sourceforge.net/
- JSR 296: Swing Application Framework
- JUIPiter:
  http://juipiter.sourceforge.net
- ReflectionBus:
  http://sourceforge.net/projects/werx/

Summary

- Design patterns and lessons from enterprise architectures should be leveraged within rich client applications
- Embed threading concerns down deep in your communication framework to limit developer mistakes
- Carefully manage painting to increase perceived performance
- Design code to scale to multiple developers using simple type-safe APIs, null object pattern, reduced use of mirrored indexed sets
- Single source of data – Render visible range of data based on query of one model rather than synching data structures between GUI components
- Try to use a mature GUI framework from the start to ease growing pains

For More Information

- NetBeans – http://www.netbeans.org
- OpenMap – http://www.openmap.org
- Foxtrot website - http://foxtrot.sourceforge.net
- Gregor Hohpe & Bobby Woolf: *Enterprise Integration Patterns*
- Brian Goetz: *Simpler, Faster, Better: Concurrency Utilities in JDKTM Software Version 5.0,* JavaOne 2006, TS-4915

Q&A

Rob Ratcliff

rob@futuretek.com

http://www.futuretek.com/ratcliff