Quality: UML diagrams

Stephen Hicks

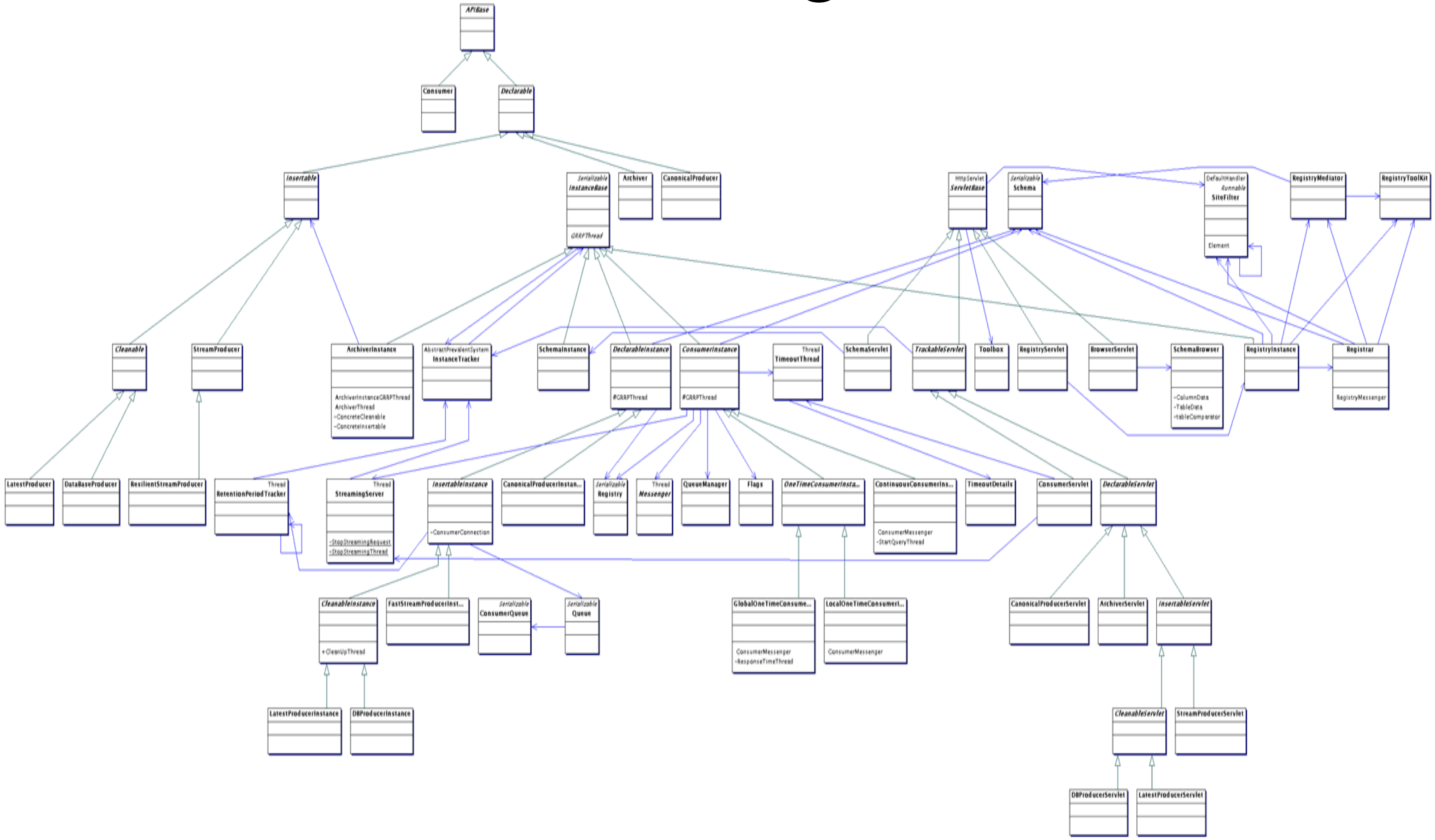
Overview

- UML diagrams
- Synchronization
- Redesign

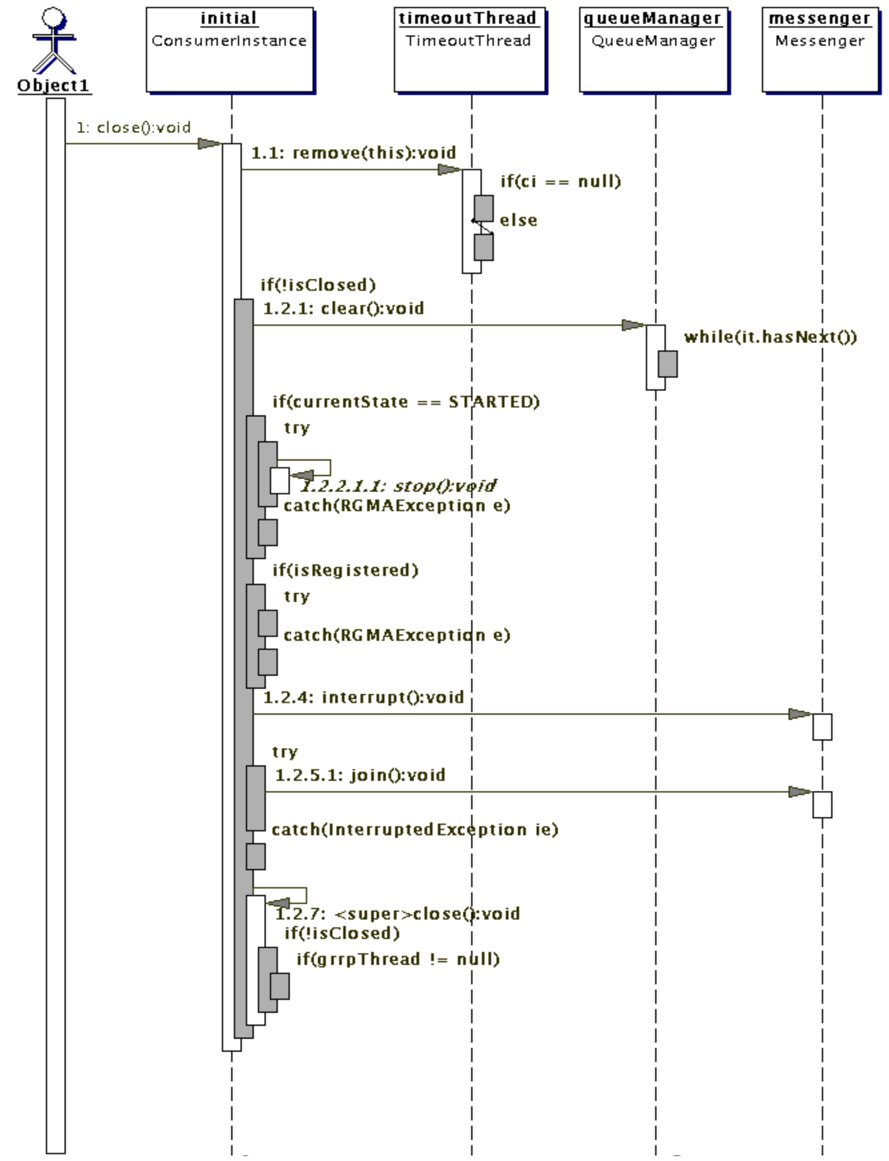
UML

- Diagrams
  - Class
  - Sequence / Collaboration
    - Synchronization problems
- Borland Together ControlCenter
  - Generate diagrams from code
    - Not enough abstraction / Too much implementation detail
    - Class
    - Sequence
      - Too detailed
      - Don't show synchronization
      - Highlight problems with code design

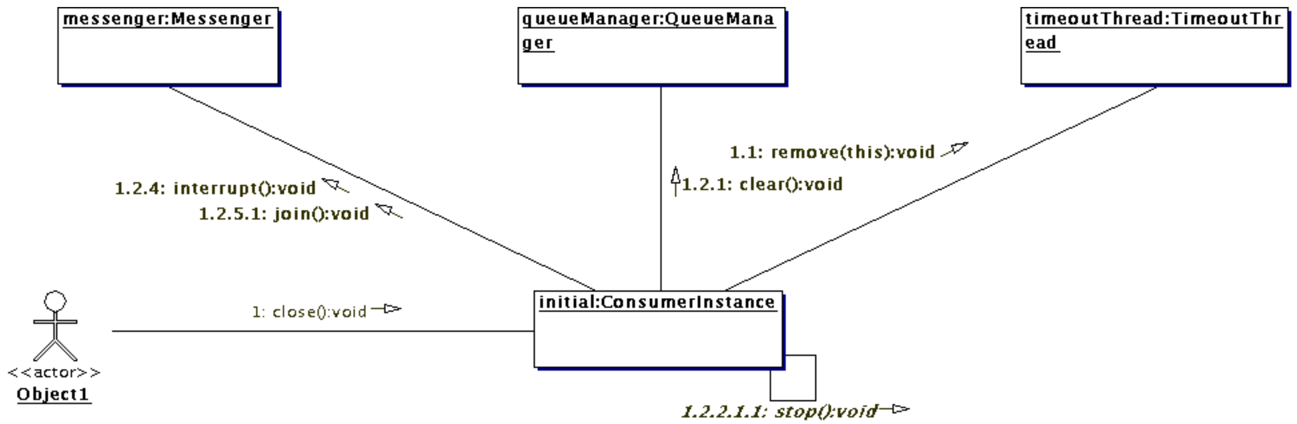
Class diagram



Sequence diagram



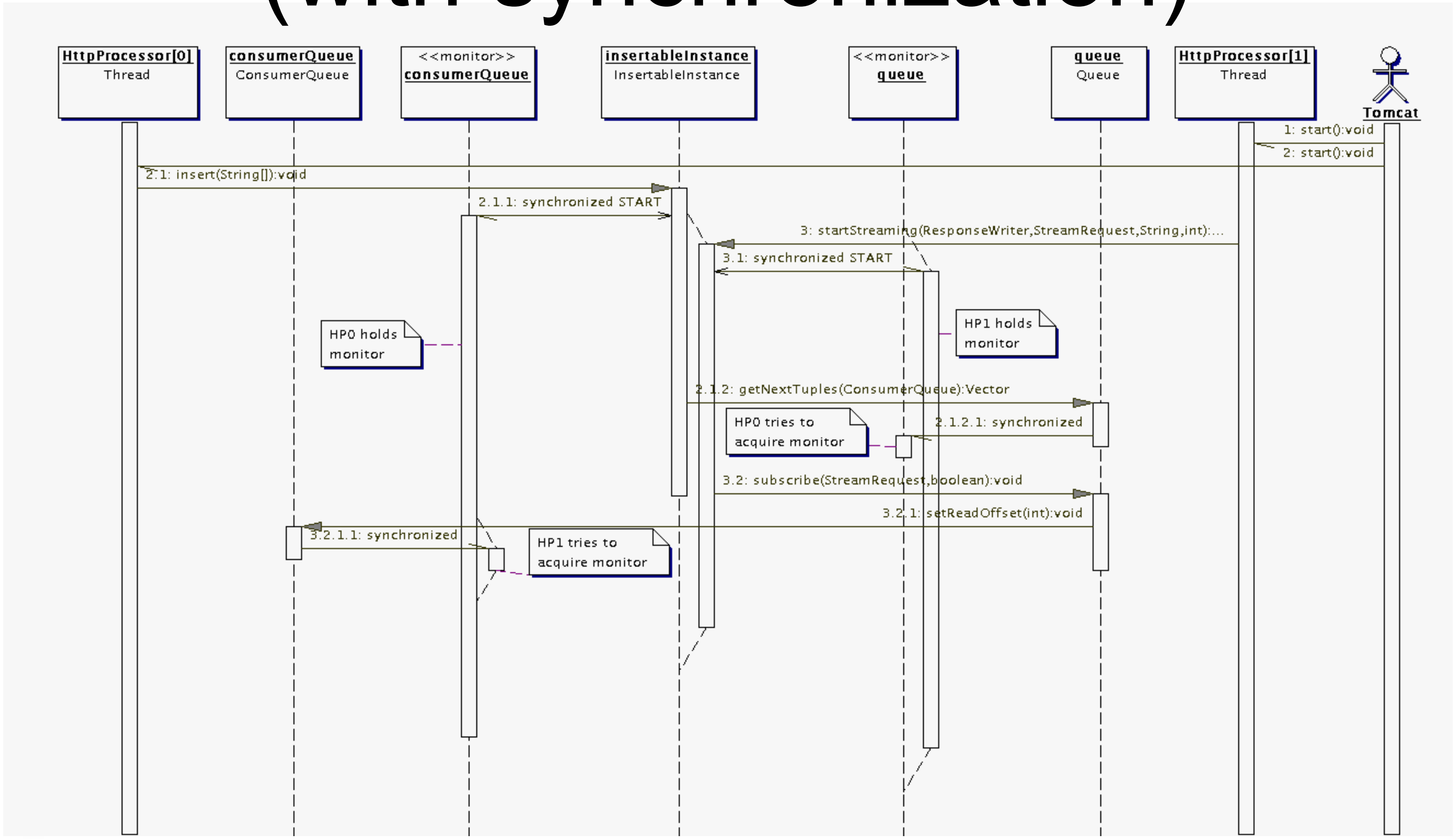
Collaboration diagram



Synchronization 1

- Deadlock problem
  - Hard to test for
  - Hard to diagnose
  - Hard to reproduce
- OptimizeIt
  - ThreadDebugger
    - Checks order of monitor acquisition
- How to visualise?
  - Sequence diagrams
    - Monitor as separate object
    - *synchronized* call to monitor shows when it is acquired
  - Collaboration diagrams

Sequence diagram (with synchronization)



Synchronization 2

- But…89 uses of *synchronized* keyword!
  - Concentrate on:
    - Threads holding more than one monitor
    - Threads holding a monitor while making a remote call
      - Messenger

Redesign 1

- In light of:
  - (Relative) code stability
  - Better understanding of the problem
  - Problems with code base:
    - Classes (and methods) are:
      - Too big
      - Not specialised enough
    - There is only **one** package!

Redesign 2: metrics

- Classes are too long
  - Main culprits are:
    - InsertableInstance (1033LOC) – Andy's refactoring?
    - TimeConverter (877)
    - BrowserSevlet (695)
  - Total LOC for R-GMA is 24385
- Classes have too many dependencies
  - Harder to test in isolation
  - Main culprits:
    - **InsertableInstance**, ConsumerInstance, ServletBase, StreamingServer, BrowserServlet, Registrar

Redesign 3

- Solution: Re(design/engineer/factor)
  - Produce UML diagrams for new system as guide for refactoring.
  - Use smaller, more focussed classes (and methods).
  - Use separate packages for producers, consumers etc.
  - Check synchronization using simplified UML model.
  - Code becomes:
    - Less complex
    - More reusable
    - Easier to instance test
    - Easier to develop in a distributed way