Implementing Parallel Algorithms
Swarm — Multithreaded Framework

continuation of Minerva project

**Basic principles:**
- multithreading should not obscure the implementation of algorithms
- a user should see the program logic, not parallelisation artifacts
- thread scheduling and balancing should be automatic

**Two levels of parallelism:**
- parallel Consumers/Producers
- Fork&Join algorithms on parallel containers

**Architecture:**
- based on the classical Producer-Consumer InfoBus pattern
- all BusMembers declare their input/output BusItem types, including possible multiplicity (one BusItem processed by several Consumers)
- pluggable Balancer orchestrates Producer/Consumer threads to optimize performance

**Design:**
- based on advanced multithreaded architecture of Java 7
- allows BusMembers in JVM-compatible multithreaded languages (Groovy, Scala, Clojure)
- possibility to re-write a part of the framework in those languages foreseen
- completely interactive with the graphical interface (various Observers)

**Future Evolution:**
- persistency (Parallel IO)
- distributed operation

**Uses:**
- Java 7
- ObjectBrowser
- Colt
- JUNG
- BeanShell
- FreeHEP
- Generic Collections
- Concurrent
- Log4J
- Groovy
- Scala
- Clojure

**Support:**
- support for multiple languages
- support for Fork & Join Containers

**Graph:**
- Bus
  - BusInterface
  - BusDefaultImpl
  - Balancer
- Interactor
  - Monitor
  - SelfTest
- AlgSupport
  - Diversity
  - Collaboration
- Object Browser
- Memory consumption
- Processing Log
- Global Operations
- Command Line
  - full Java available
- Dynamical Graphical view of
  - Producers/Consumers/Algorithms
  - Busitems
  - calls

- Continuation of Minerva project