

# Problem Solving Environments and Legacy Code Integration with Java

---

Omer F. Rana  
Department of Computer Science  
Cardiff University, UK  
[o.f.rana@cs.cf.ac.uk](mailto:o.f.rana@cs.cf.ac.uk)

# Themes

- What are PSEs
- Tools for PSEs and Applications
- Benefits and Pitfalls
- Resource Discovery in PSEs
- Taking Stock: Why Java is crucial in PSE infrastructure

# Scientific Computing = HPC, ++-

- Best abstraction + best performance?
- What do the users' want? Not necessarily performance. Target community? Ease of use?
- HPC Apps
  - ◆ AOL (1000s of CPUs with 1000s of users)
  - ◆ Embedded and mobile computing (millions of devices, millions of users)
  - ◆ Large scale vs. high performance
  - ◆ high throughput vs. high performance

# Why PSEs?

- *Need*
  - ◆ enhanced scientific insight; reduced development costs; improved product quality and industrial efficiency.
- *Need*
  - ◆ transparent means of integrating distributed computers, instruments, sensors, and people.
- *Need*
  - ◆ improved software productivity to extract maximum benefit from advances in computers, networks, and algorithms.

# Making it easier!

- Configuring machines is hard - every app may require a specialised environment
- Programming models are not intuitive or hard to understand
- Although tools are available - usability is low - why?

# Driven by: Who Uses PSEs

- Infrastructure should support two types of users
- Application Scientists/Engineers
  - ◆ solve a particular problem which is domain specific
  - ◆ undertake "what if" investigations
- Developers (Programmers)
  - ◆ provide specialised solvers, editors
  - ◆ provide schedulers, resource managers

# Use Cases ... 1/3

- Remote use of an existing solver
  - ◆ Local data set
  - ◆ Licensing constraints on running solver
  - ◆ Hard to manage local configuration
  - ◆ Visualising results of a remote solver (graphical output, or interactive steering)

• *Parameter runs on a given solver*

# Use Cases ... 2/3

- **Sharing data from multiple experiments**
  - ◆ Writing output of a simulation into a database
  - ◆ Annotate results of an experiment
  - ◆ Correlate multiple runs of an experiment
  - ◆ Support data management for each component

- 
- *Migrating Large Data sets across a network*
  - *Data distribution between components*

# Use Cases ... 3/3

- On-demand use of executable codes
  - ◆ Execution environment can support different architectures
  - ◆ Solvers can be employed on-demand, based on data flow identified by a user
  - ◆ Resource scheduling undertaken locally by resource managers

# Motivation

- Construct scientific applications, visually, from Components - the infrastructure we call a Problem Solving Environment (PSE)
- Software Engineering for Scientific Computing
- Components can be written in a number of languages: Java, Fortran, C -- may be binary or sources
- Components can be sequential codes, parallel (MPI or PVM based) -- could be sub-routines or complete applications
  - ◆ hence, component granularity can vary

# Many Types of PSEs

- Mathematical
  - ◆ MatLab, Mathematica, Maple
- Component based
  - ◆ ARCADE, WebFlow/Gateway, ADVICE
- Problem Specific
  - ◆ ECCE (Computational Chemistry), Cactus (Gravitational Waves)
- *Can we find common themes?*

# Yes! - Common Themes

- MPSEs = PSE Infrastructure + Application Specific Knowledge Integration
- Application Construction Tools
  - ◆ Visual Programming Tools
  - ◆ Component Repositories
- Application Execution Tools
  - ◆ Schedulers and Resource Managers
  - ◆ Meta Computing Environments

# Impacts

## Problem Description/Evaluation Tools

- Visual Prog., Iris Explorer, MatLab, ADL(Erlangen-Nuremberg)

## Component Frameworks

- Imperial College, ECA, INRIA, Gateway, Arcade, Advice etc

- ## Legacy Codes (wrappers)
- SWIG, WG

## PSE Infrastructure

## Resource Management

- Codine, LSF
- Globus, Legion, Condor

Learning

Visualisation

Steering

## PSE Services

Persistence

Data Management

Result Sharing (Electronic Notebooks)

Sharability

Repeatability

## Usage

Usability

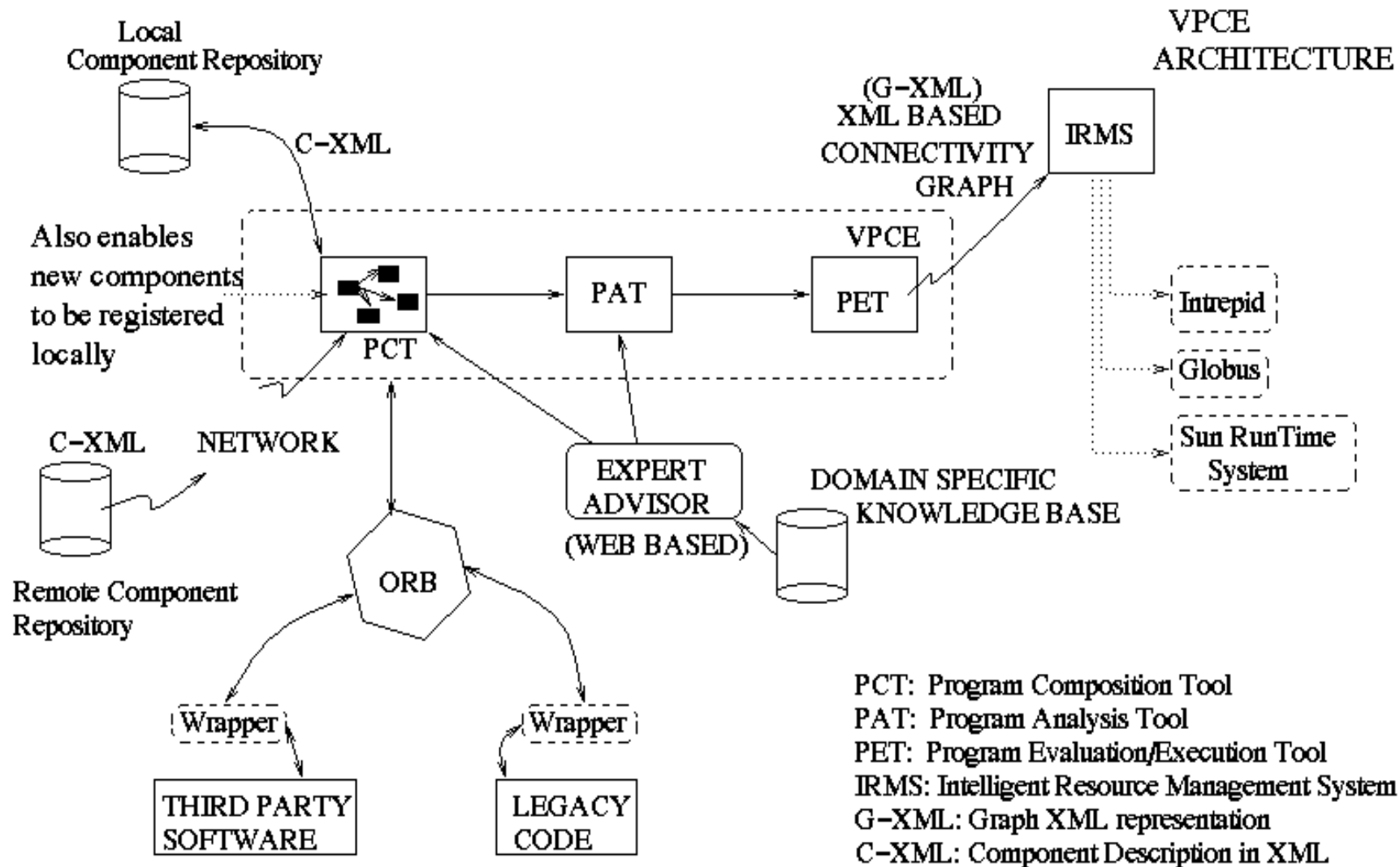
Maturity

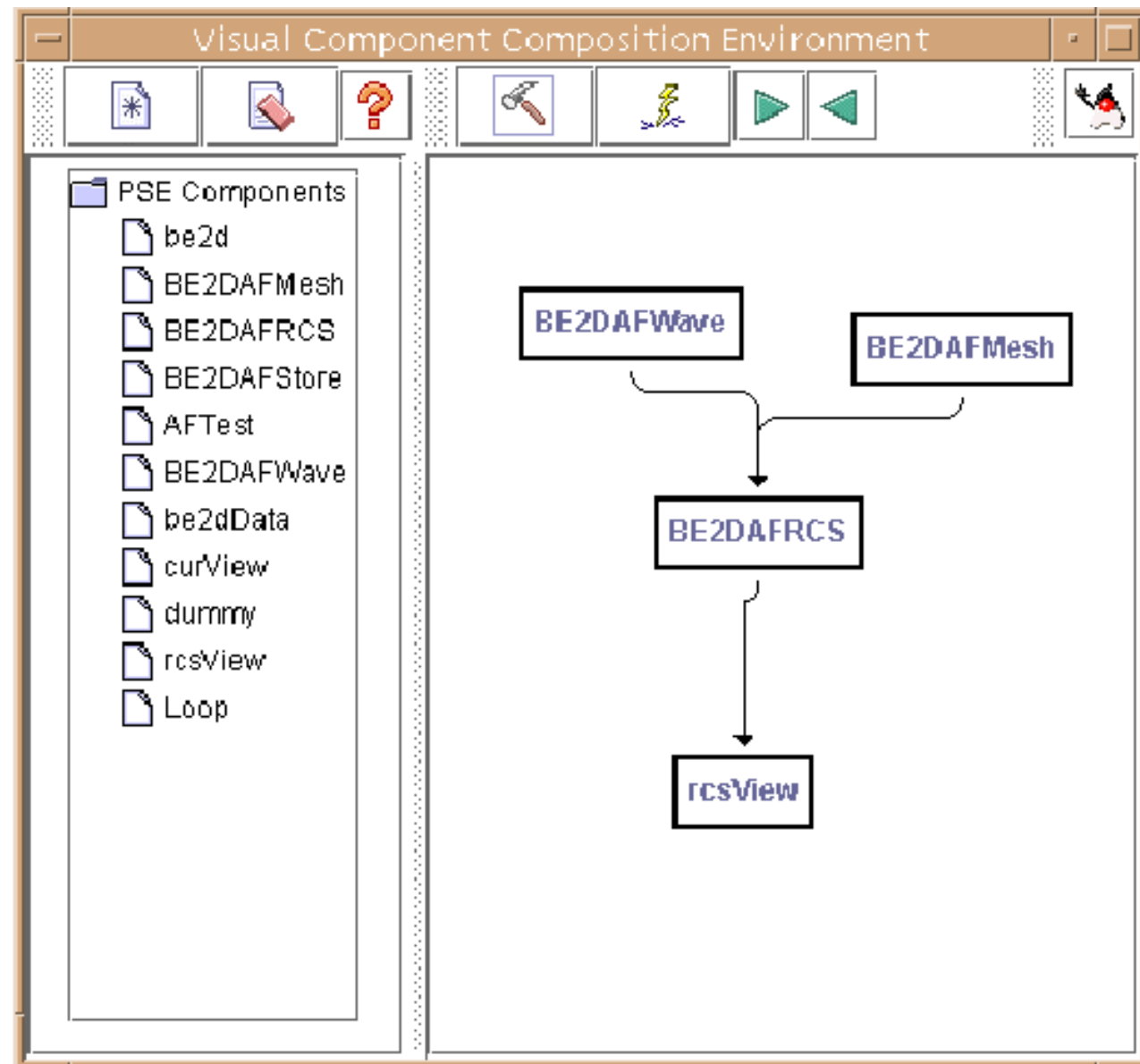
ALL Three are important

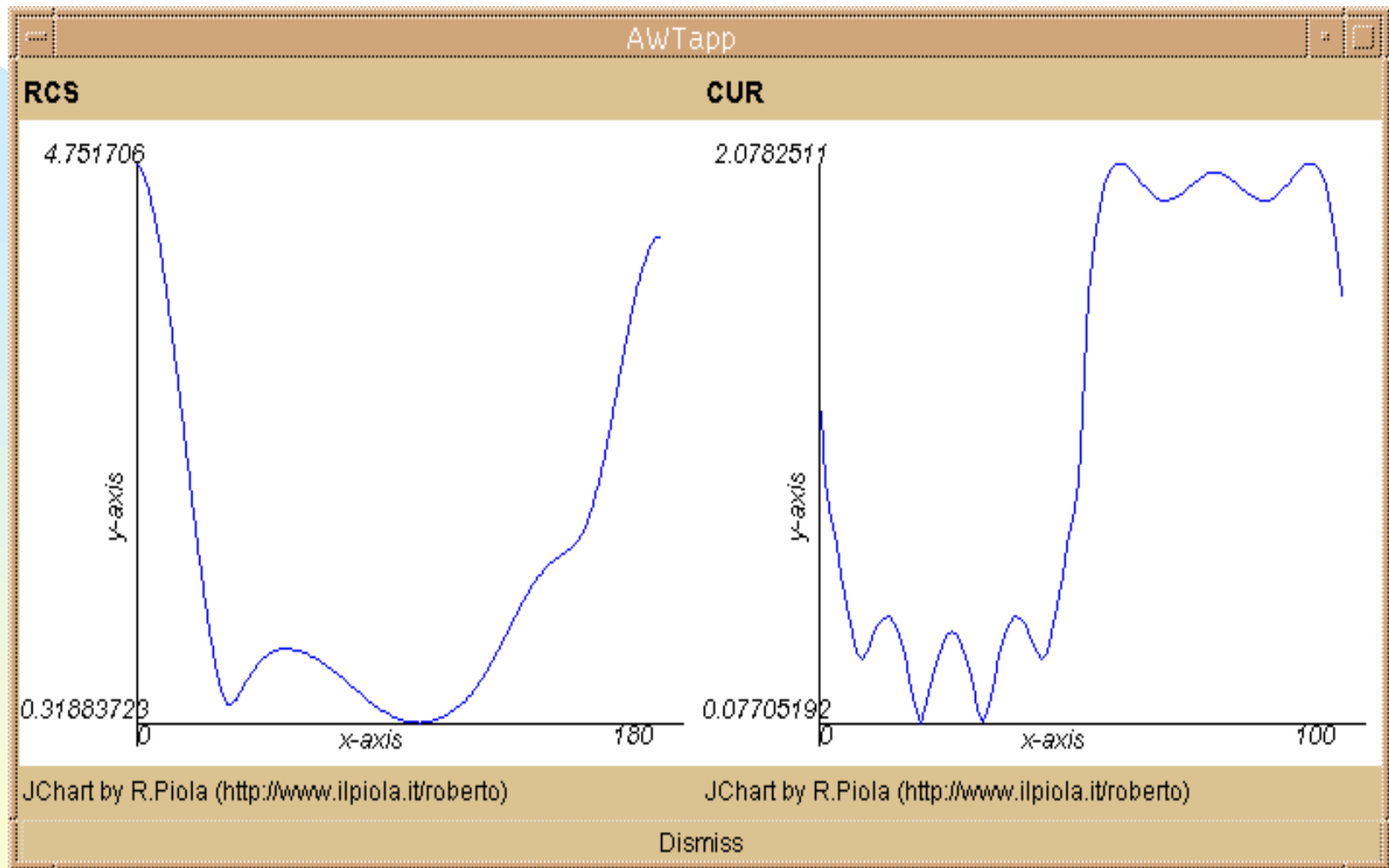
UKHEC

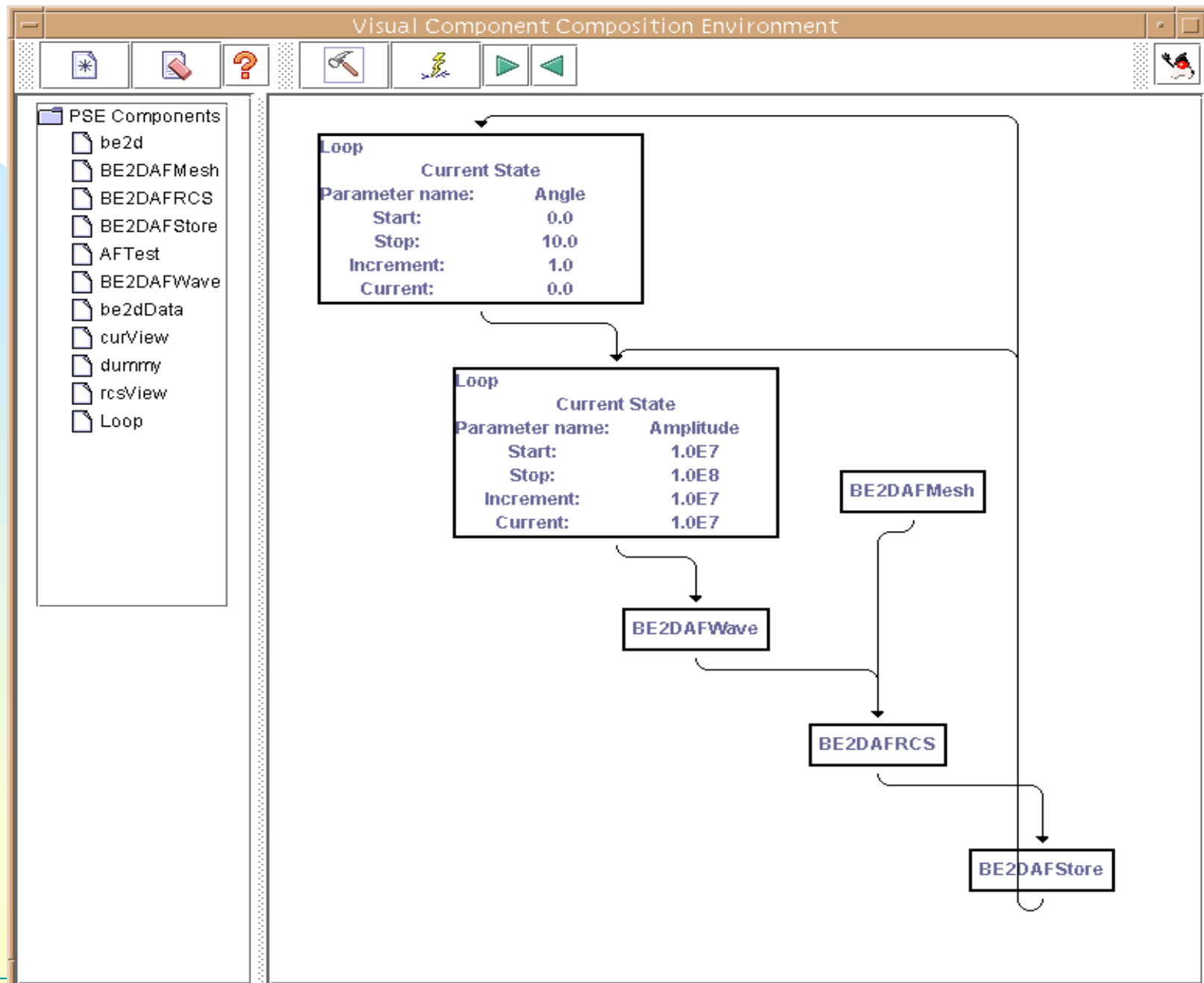
RDIFF UNIVERSITY

# Architecture

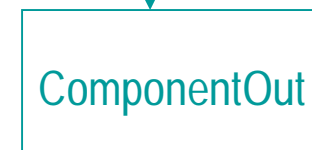
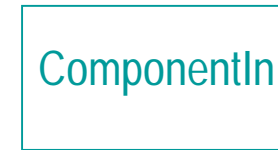
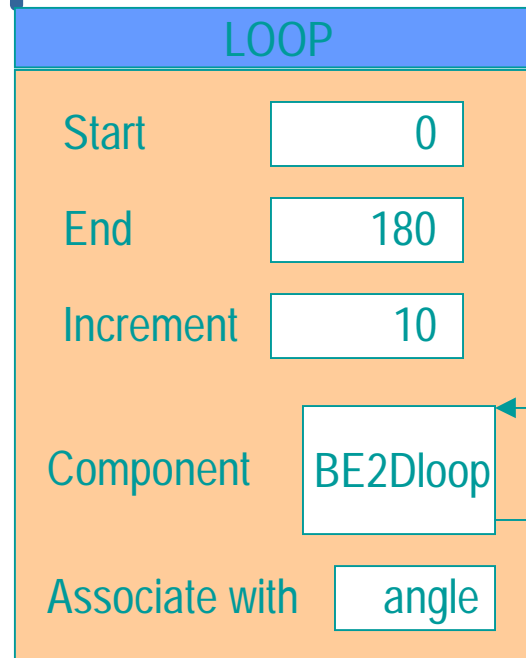
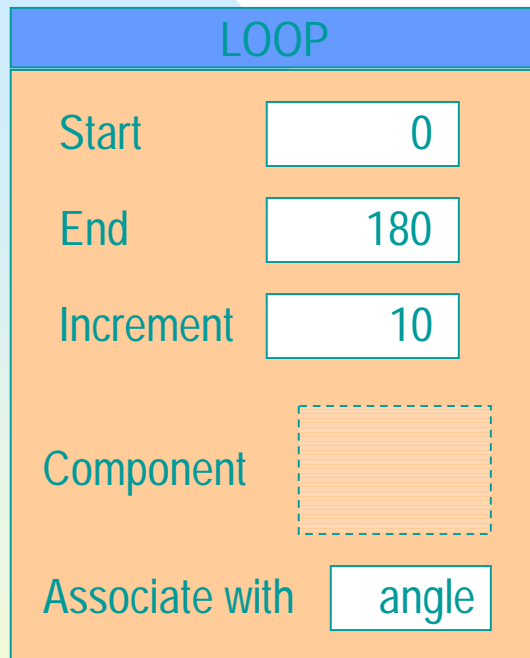








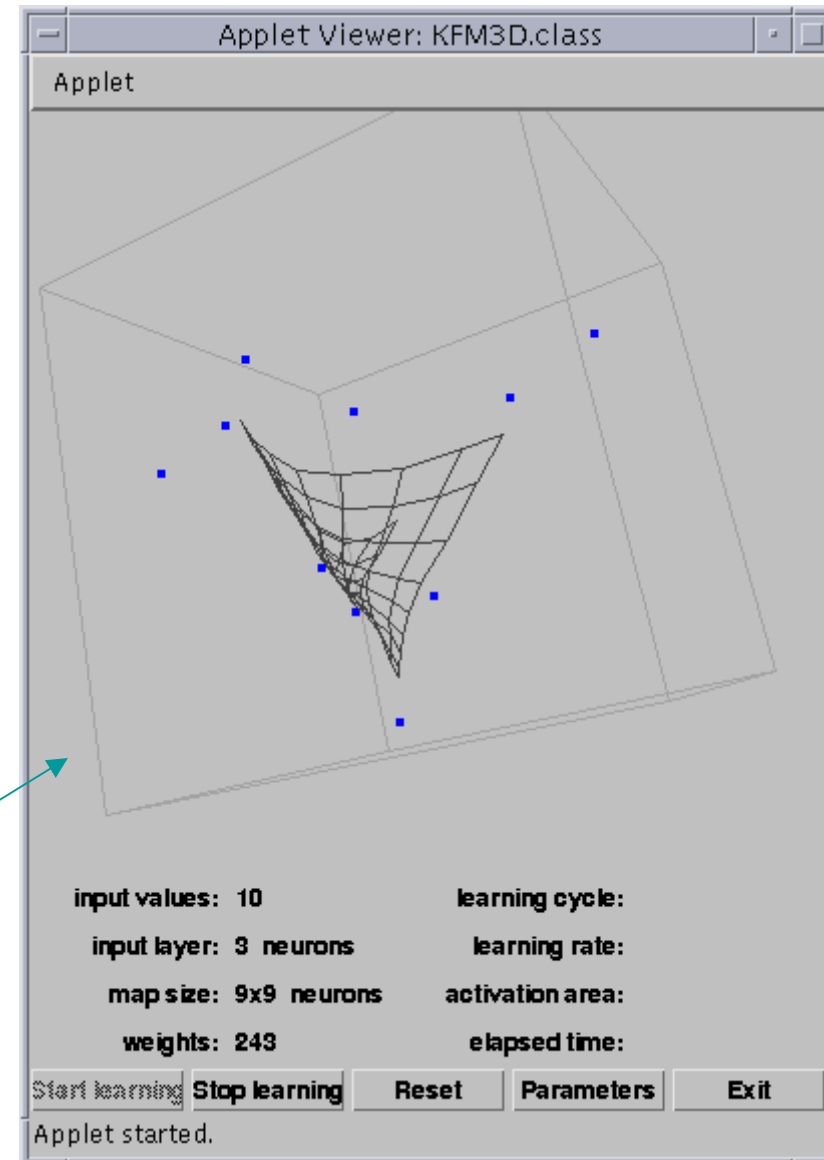
# The Loop Control Template



- Assume independent loops.
- Output from each loop goes to ComponentOut
- Termination of loop triggers an event that tells ComponentOut that loop is done.

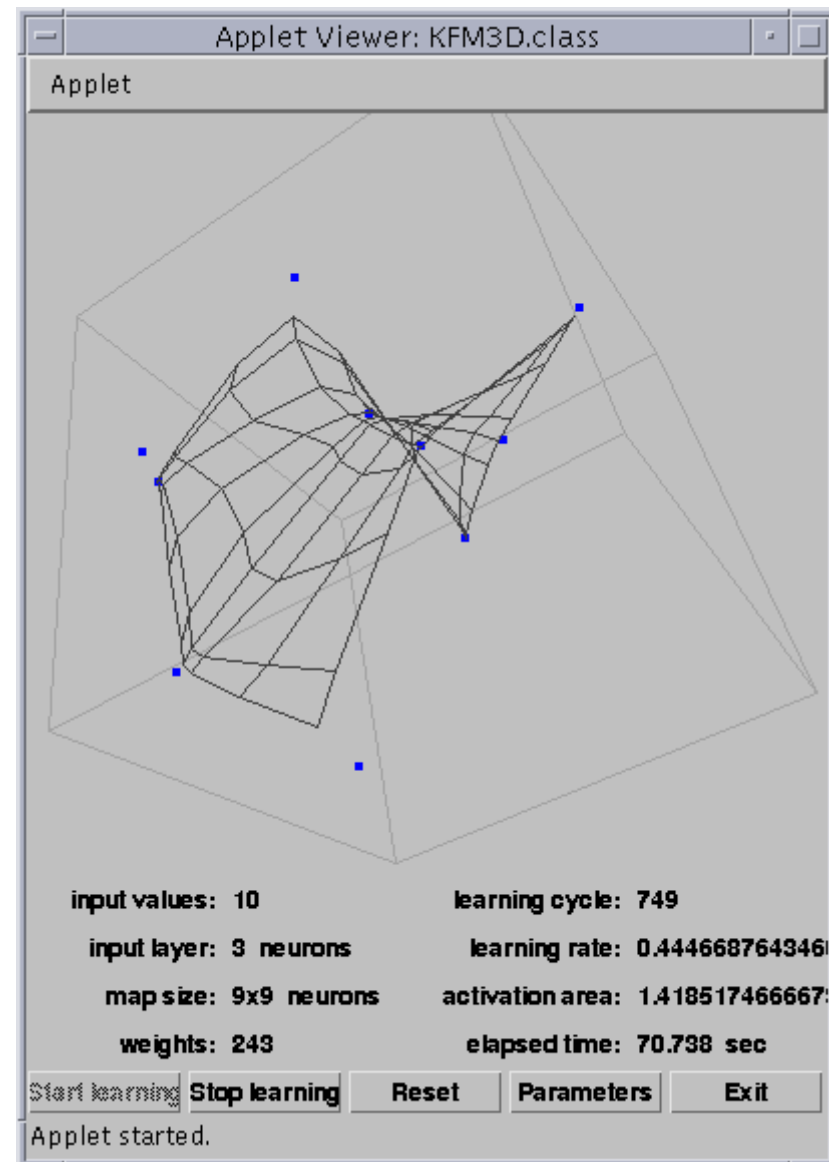
# Visualisation

- Neural network trying to fit a surface to a set of data points
- SOM example



# Visualisation

- Neural Network training close to being complete



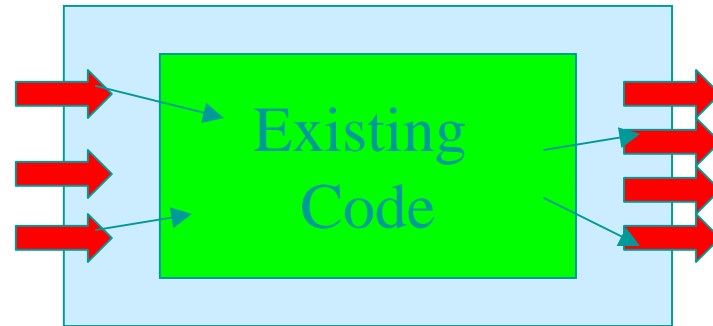
# Show XML Component Model

# Component Model and Extensions

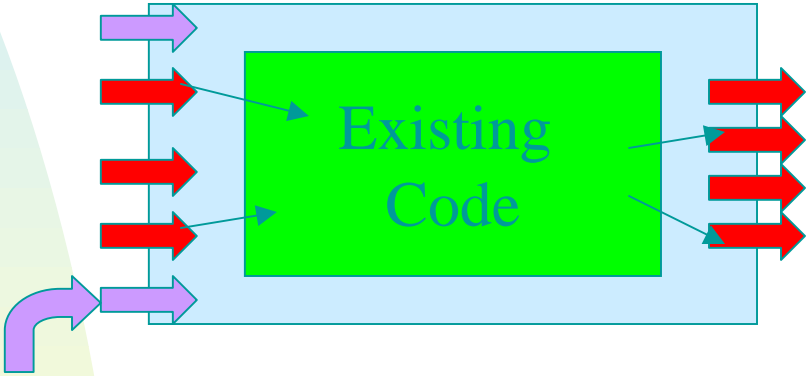


Existing  
Code

# Component Model and Extensions

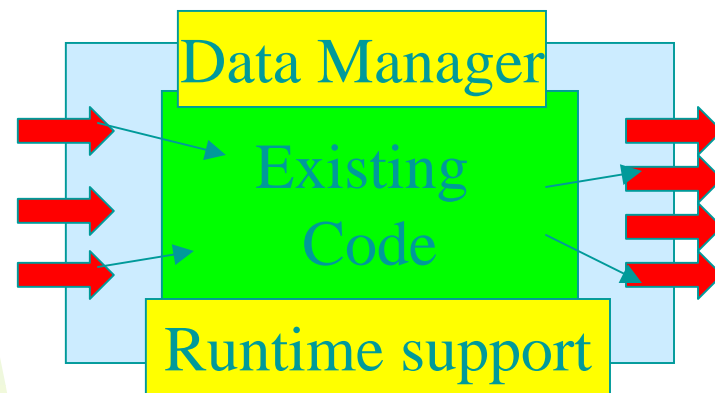


# Component Model and Extensions

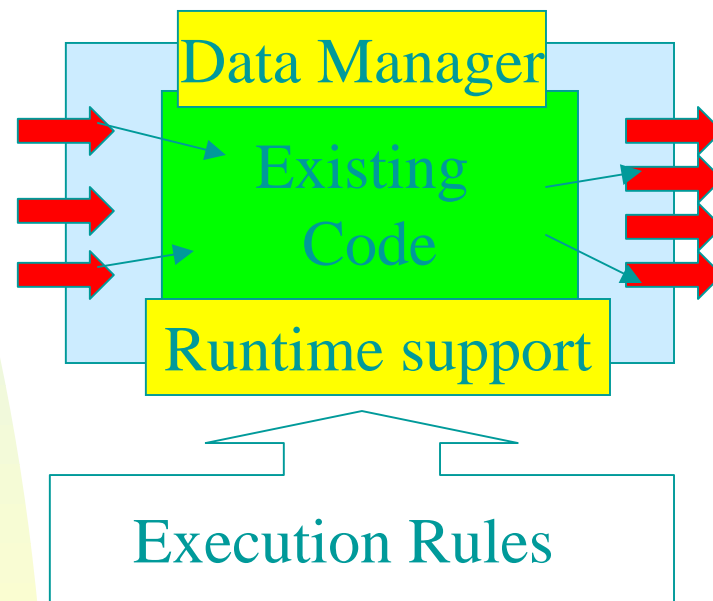


*External Control  
Input (for Steering)*

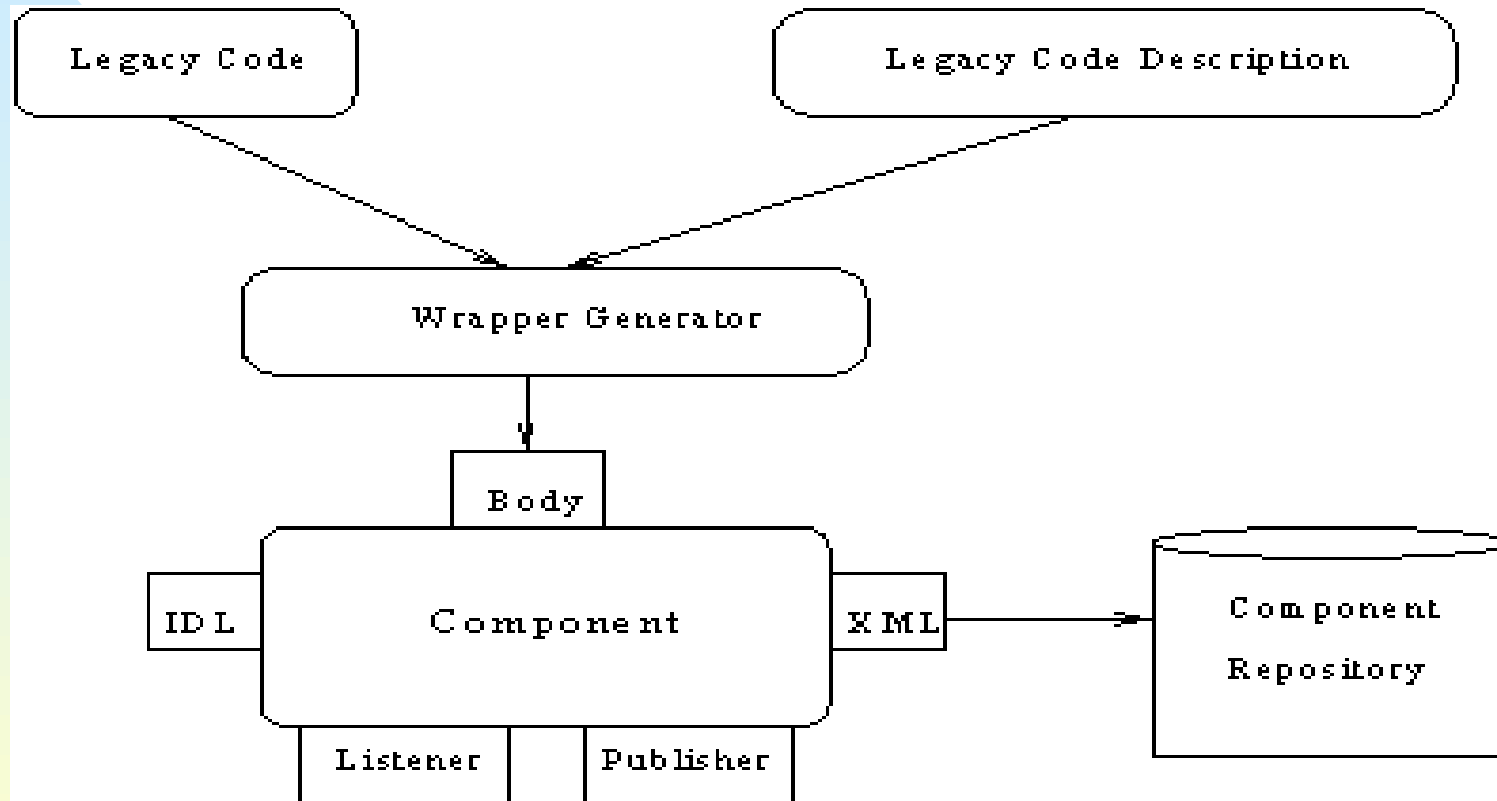
# Component Model and Extensions

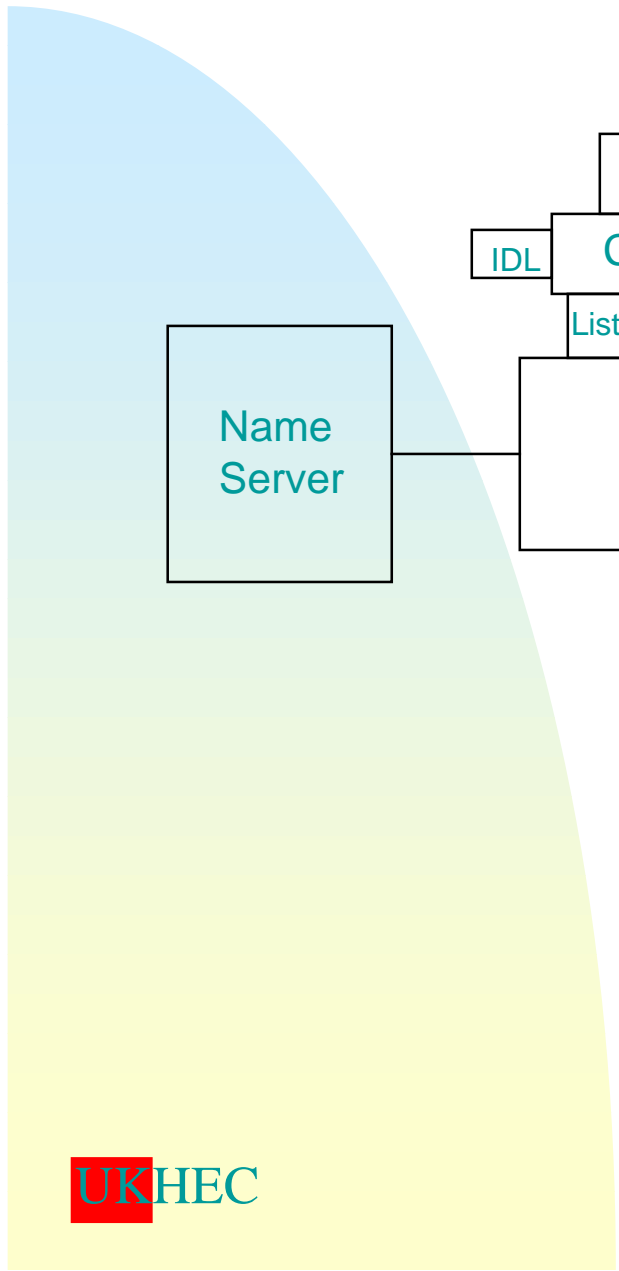
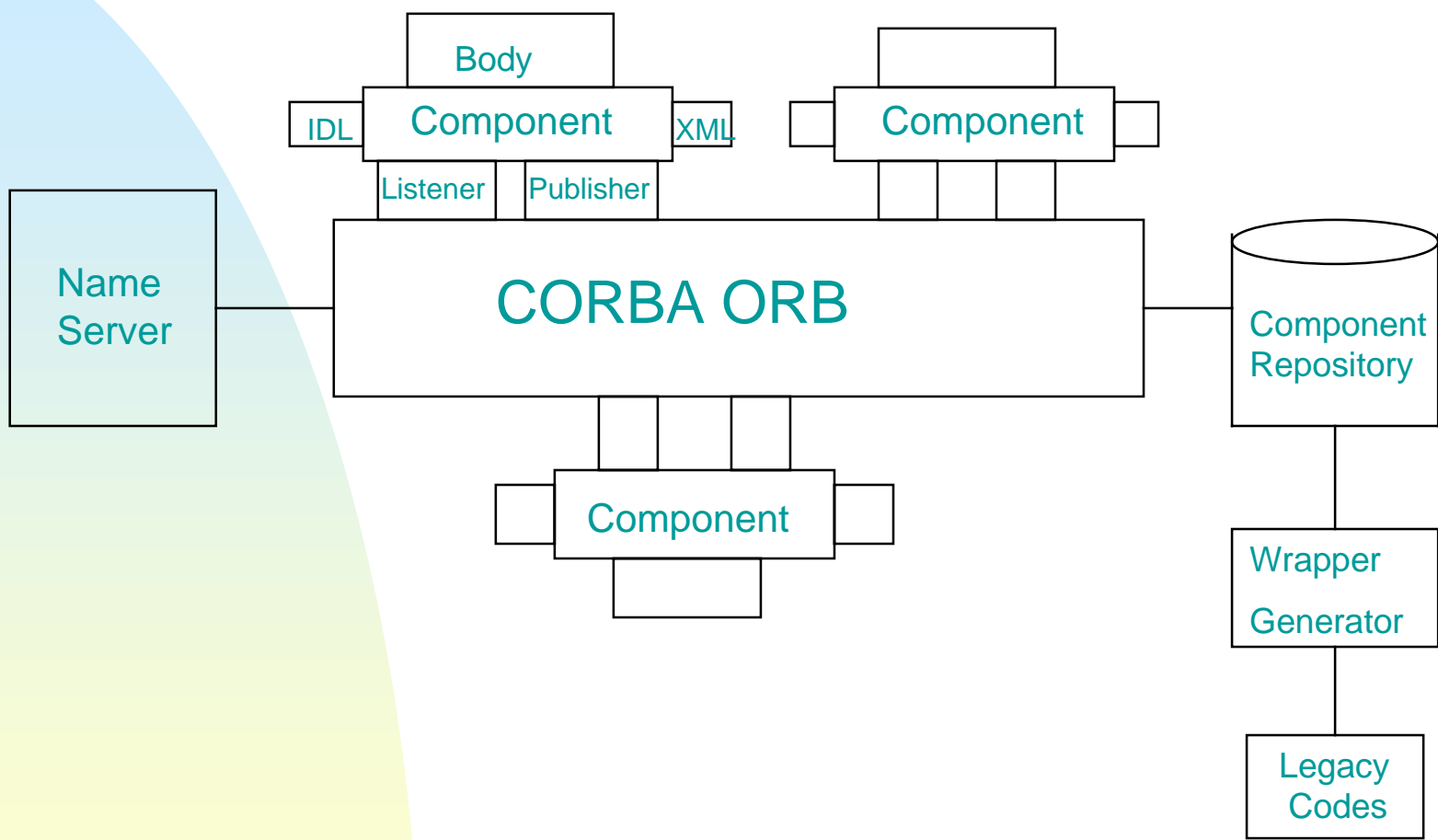


# Component Model and Extensions



# Operations in the WG





# An Application using the WG

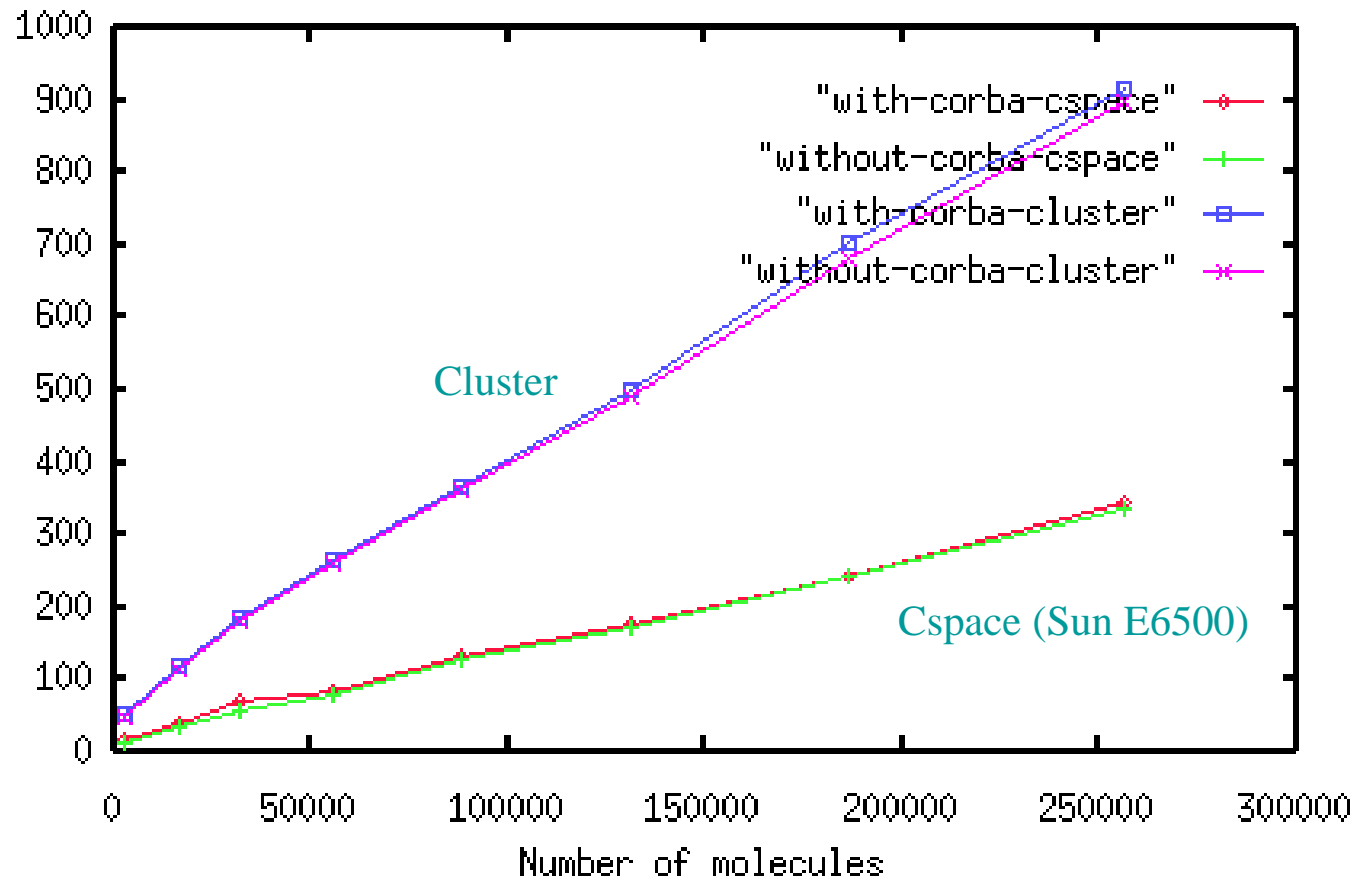
- The legacy code-- A legacy code for molecular dynamics simulations written in C using MPI
  - ◆ Based on the Lennard-Jones Fluid
  - ◆ Performs Force and Velocity calculations
- Code wrapping in two ways (Use Visibroker/Java ORB 4.2 from Inprise)
  - ◆ Single Object
  - ◆ Multiple Objects

# Performance Evaluation

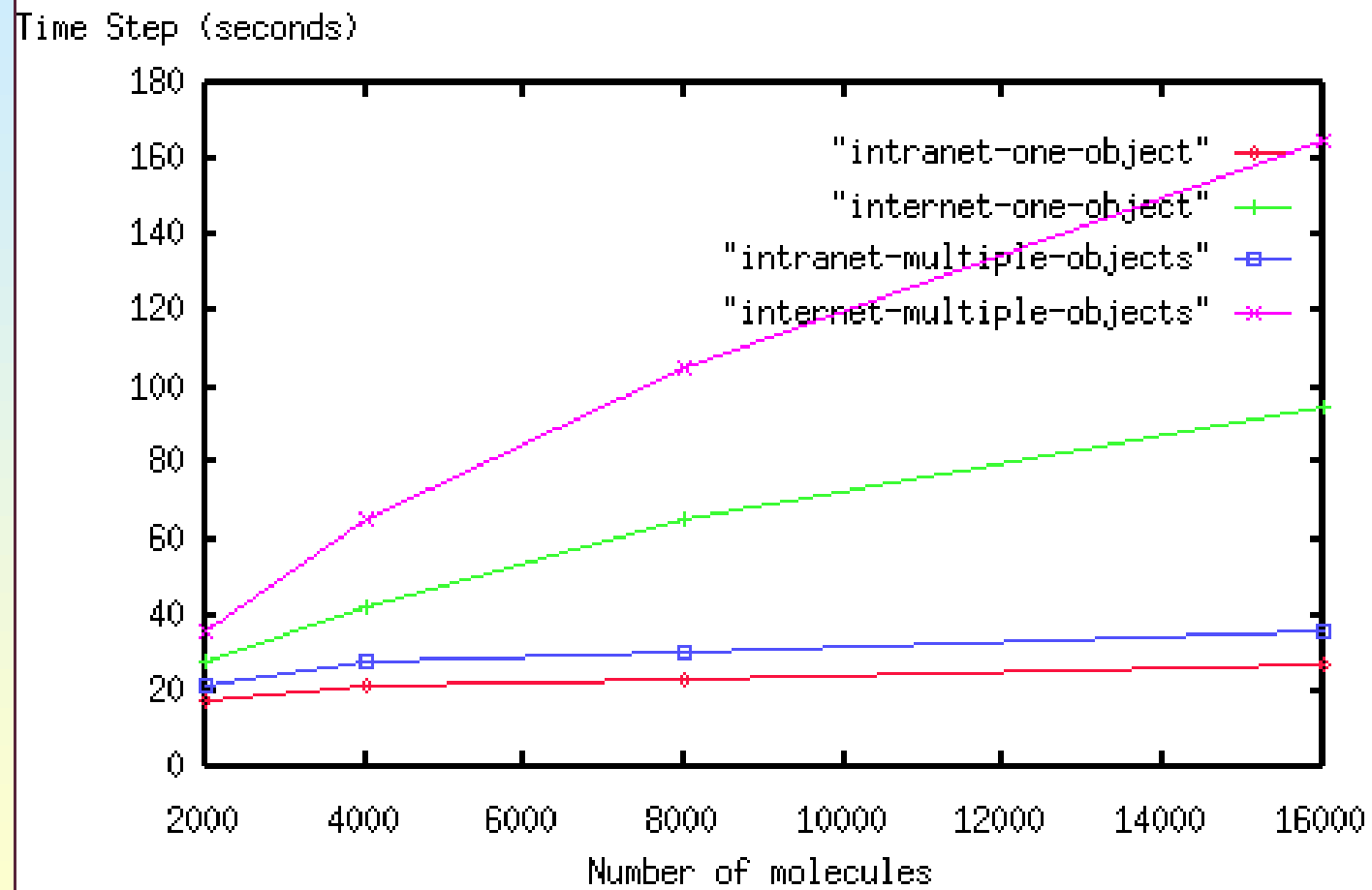
- Experimental Environments
  - ◆ Running the legacy code itself and the wrapped CORBA object respectively on a cluster of workstations and a parallel machine.
  - ◆ The number of molecules is increased from 2048 to 256,000.
  - ◆ A cluster of workstations running Sun Solaris2.7 and MPICH1.2.0, connected over an intranet (with shared file space) with 10Mb/s Ethernet.
  - ◆ The parallel machine is a Sun E6500 with thirty 336MHz Ultra Sparc II processors, running Solaris2.7 and using MPI libraries from Sun Microsystems.
  - ◆ Using 8 workstations in the cluster and 8 processors of the parallel machine

# Performance

Execution time (seconds) for 500 time steps



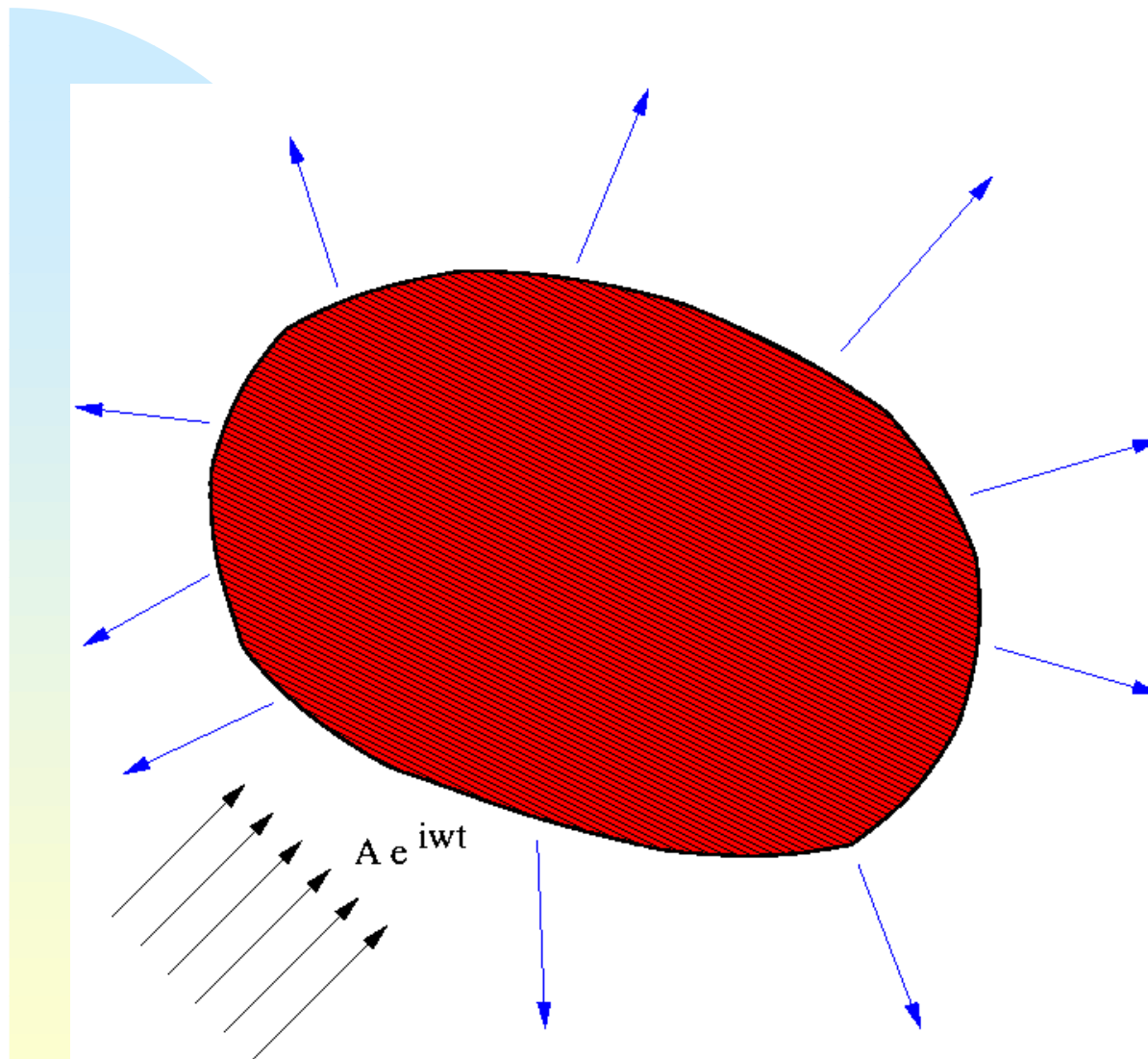
# Performance



# A VCCE for Computational Electromagnetics

- We have implemented a VCCE and prototype PSE for a 2-D computational electromagnetics code, BE2D.
- The boundary element method is used to solve Maxwell's equations in two dimensions.
- Input is:
  - ◆ contour of  $(x,y)$  values defining the boundary of the 2-D surface
  - ◆ angle of incoming wave relative to surface
  - ◆ frequency of incoming wave
  - ◆ amplitude of incoming wave
- Output is:
  - ◆ file of radar cross section (RCS) data
  - ◆ file of surface current data

# EM Scattering Problem



Maxwell's equations



Dense linear system



Surface current

# Solution Outline

- A mesh generation component outputs a set of  $(x,y)$  coordinates specifying the boundary of the surface. This is in the form of a CORBA data object.
- A wave control component specifies the angle, frequency, and amplitude of the incident wave.
- The BE2D boundary element solver accepts input from the mesh generation and wave control components, and outputs the scattering prediction and surface current as CORBA data objects.

# Solution Details

- The solver is used for the analysis of electromagnetic-wave scattering from infinite cylinders, with the profile in the x-y plane and the cylinder axis along the z-axis.
- The grid required for the solver consists of a list of (x,y) coordinates around the boundary of the scatterer. Analytical shapes are often used as the basis of the scatterer profile.
- The code computes the solution of the EFIE (Electric Field Integral Equation) form of the Maxwell equations for Transverse Electromagnetic (TE) waves using a Method of Moments technique.
- The equations are discretised using a Galerkin scheme of 2D Rao-Wilton-Glisson linear elements positioned along the boundary contour of the cylinder.
- A dense matrix is computed and solved with a LU solver, producing a complex current distribution around the boundary.

# How It Works

- An XML file is used to specify the interface of each component, and other details such as
  - ◆ where the component executable resides
  - ◆ the URL of a help information on the component
  - ◆ performance-related information
  - ◆ constraints on the execution of the component
- The VCCE uses the XML file to populate the Component Repository
- A special Java/CORBA server object called the *ActionFactory* controls the creation and execution of the components. This is based on the Abstract Factory design pattern, and was written by Dassault Aviation within the JULIUS Project.
- The third-party *JChart* software can be used within the VCCE to display results graphically.

# Participants

- Cardiff University
  - ◆ Matthew Shields, Maozhen Li
- Southampton University
  - ◆ Tony Hey, Jeff Reeve, Dave Lancaster, Simon Cox
- IT Innovations (formerly PAC)
  - ◆ Matthew Addis
- Oak Ridge National Lab
  - ◆ David Walker
- BAe Systems
  - ◆ David Golby

# Benefits

- Ease of use overrides high performance
- components enable sharing of common services across disciplines
- Expert Advisor can constrain possible alternatives
- Can be used as a front end to emerging projects: Grid Portals
- Visualisation can be easily added
- Can integrate many different styles: MatLab (end-to-end), GAMS, NAG

# Pitfalls

- Too many expectations - should not oversell
- Involve end users early on - computer scientists (including myself) get carried away
- Usability surveys are essential, and should be undertaken early in the project
- May not be useful/beneficial in all situations - use for both low and high end services
- Wrapping legacy codes is not easy: primitive vs. derived types, numerical accuracy
- Sometimes performance IS everything!

## Pitfalls/Benefits ... 2

- Physicists may not be the best audience - look to biology, engineering, finance
- Not too abstract - otherwise no users
- Not too trivial - otherwise no users
- First NSF workshop in 1994 - need to evaluate priorities once again:
  - ◆ Problem description
  - ◆ Resource Management

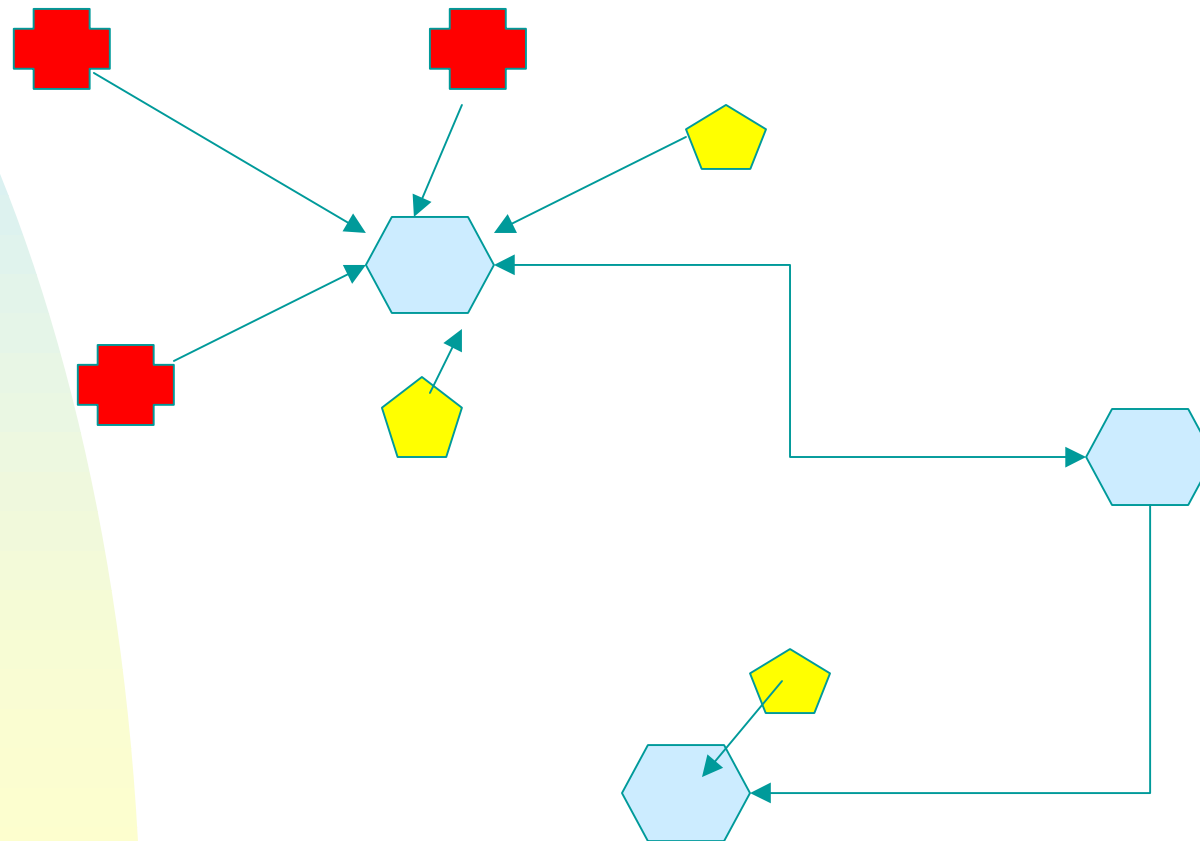
# Future Directions ...

- Are we just re-labeling technology, or is there something else?
- PSE lifecycle - influenced by Software Engineering approaches (Capability Maturity Model)
- Evaluating PSE projects - how valid is the approach? How do the users respond?
- Component types vs. Component behaviours (enable user to select)
- PSE projects: terascale facilities, to PSEs in education (both are important)

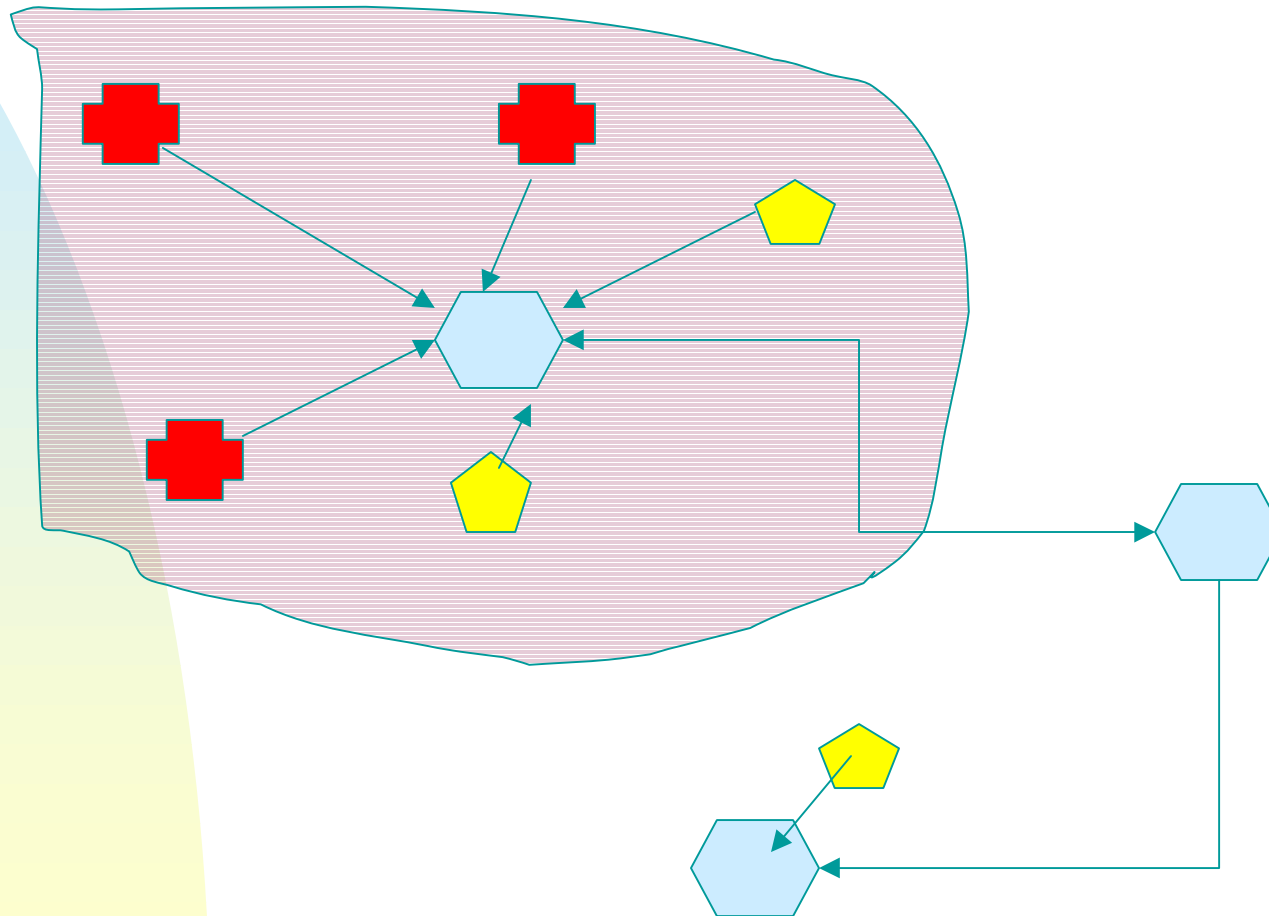
# Resource Discovery

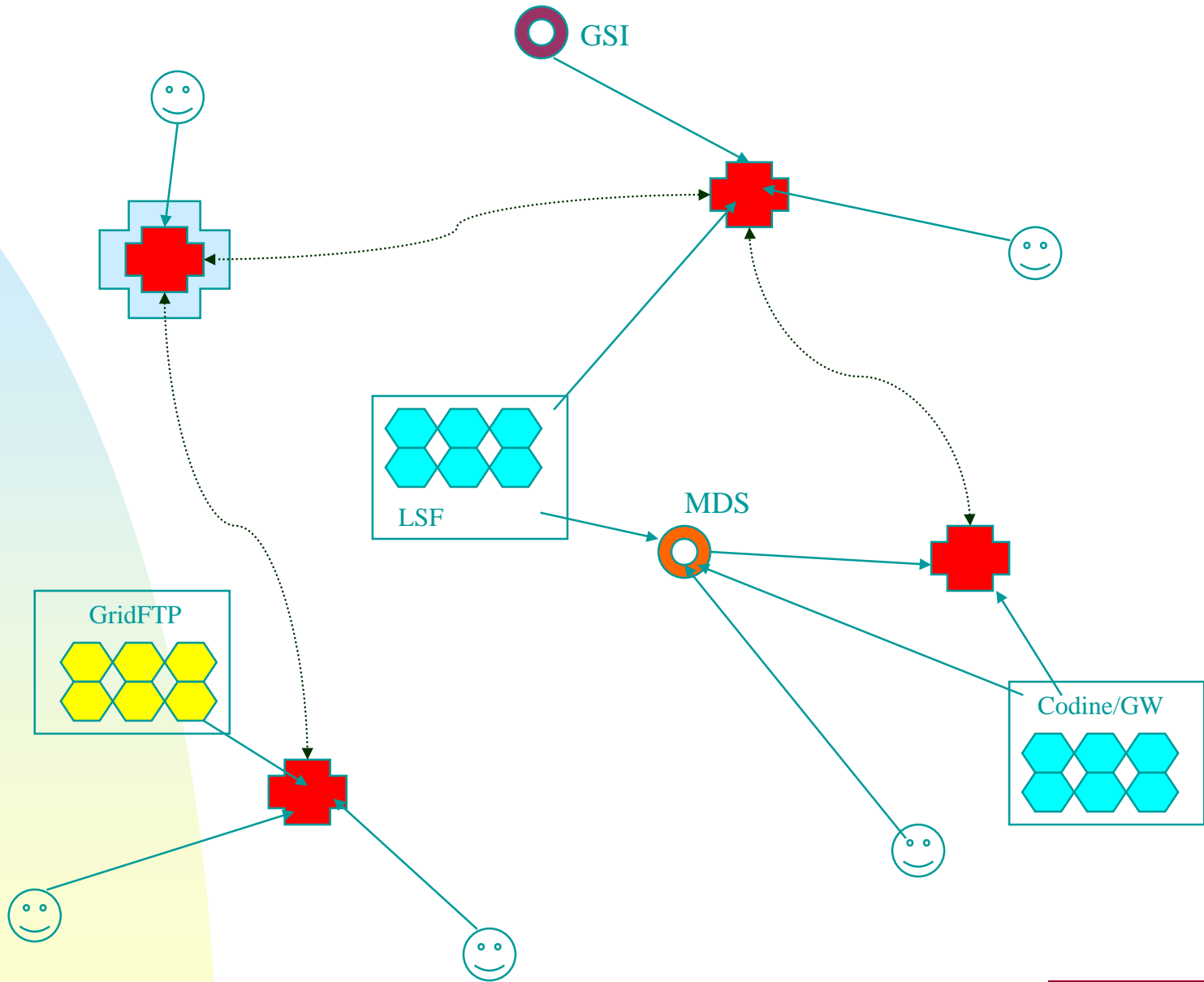
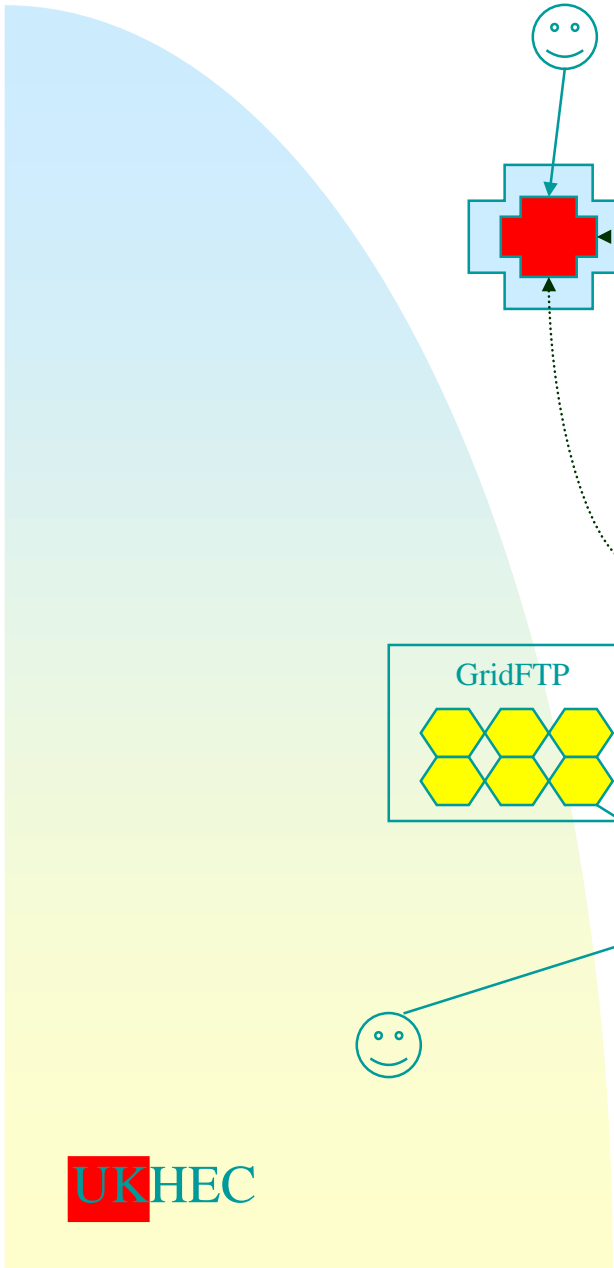
- Devices register with a resource broker
- Resource broker identifies device capabilities
- Tasks request capabilities, based on their requirements
- A MatchMaker matches resource requests with task capabilities
- Based on "class ads" in CONDOR

# Resource Discovery



# Resource Discovery





# Use of Java

- Use of Java is crucial in PSE infrastructure
  - ◆ Component Wrapping (with some implementations very good perf.)
  - ◆ Support for User Interaction
  - ◆ Glue for integrating codes from different users
  - ◆ Enable each component to support its own runtime (`Java.Runtime.exec()`)

## Use of Java ... 2

- Resource discovery support
  - ◆ Name space management
  - ◆ XML and similar metadata tools available
  - ◆ Availability of some numerical libraries and parallel libraries (MPI, JavaNumerics, Matrix package)
  - ◆ Jini

# To find out more ...

<http://www.cs.cf.ac.uk/Pseweb/>

- Architecture of a distributed PSE (Concurrency: P&E, March 2001)
- Visual Program Composition Environment for a PSE (Concurrency: P&E, September 2000)
- Wrapping Legacy Codes as Java/CORBA components (SC2000 and HPDC9 (AMS Workshop), 2000)
- Agent based Resource Integration in PSE (16th IMACS Symp, August 2000)
- Parallel and Distributed Data Mining Application Suite (IPDPS/SPDP, May 2000)
- Component models for PSEs in XML (GCSE, Erfurt, Germany, September 1999)