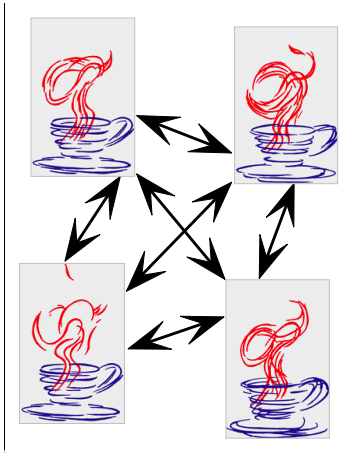


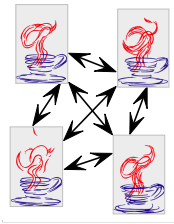
Programming Metasystems with Active Objects

M. Di Santo, F. Frattolillo, N. Ranaldo, E. Zimeo

*University of Sannio – Benevento - **Italy***

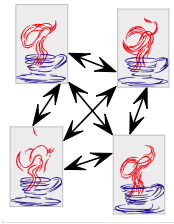
W. Russo

*University of Calabria – Rende (CS) - **Italy***



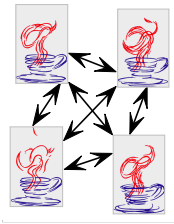
Contents

- Introduction and motivations
- Models for programming metasystems
- A brief introducton to HiMM
  - The *Hierarchical Metacomputer* architecture (HiM)
  - The *Customizable* architecture of a Node
  - The API of HiMM
- The integration of ProActive with HiMM
  - Performance analisys
- Conclusions and future work



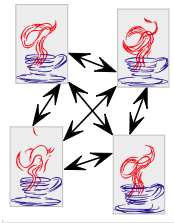
Introduction and motivations

- The presence of a huge amount of computers and mobile devices results in an increased focus on the interconnection of systems
- This evolution of the computer scenario has promoted two new trends in distributed computing:
  - **Grid computing**, for scientific applications
  - **Web Services**, for e-commerce and business applications
- In both domains, it is important to
  - define suitable programming models in order to better exploit large-scale distributed systems
  - improve customizability of middleware platforms



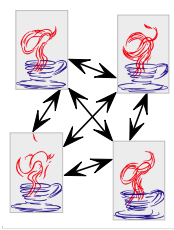
Models for programming metasystem

- Most of the middleware platforms for Grid computing does not define a model for programming parallel applications
- The send/receive model is used as the de-facto standard for communication both in cluster environments and in large heterogeneous distributed systems
- However, to handle the unpredictability of resource avalibility and behavior more dynamic programming models are required
  - Agents
    - completely change the way distributed applications are designed and deployed
  - Active Objects
    - allow for taking advantage of well-defined theories and design techniques



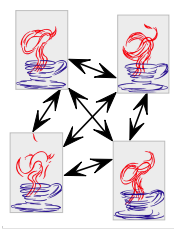
Our proposal (1)

- Even though many active objects libraries have been developed, the diffusion of Grids has introduced new issues to be taken into account during program development
  - heterogeneity, scalability, unpredictability and adaptability
- We think that the separation of aspects gives the proper flexibility for programming Grid applications
  - computational entities should exploit communication, security and management features of a middleware platform without affecting the functional aspects of an application
- So, we have chosen an approach based on
  - A generalized, customizable middleware platform
  - A customization to support an active object programming model



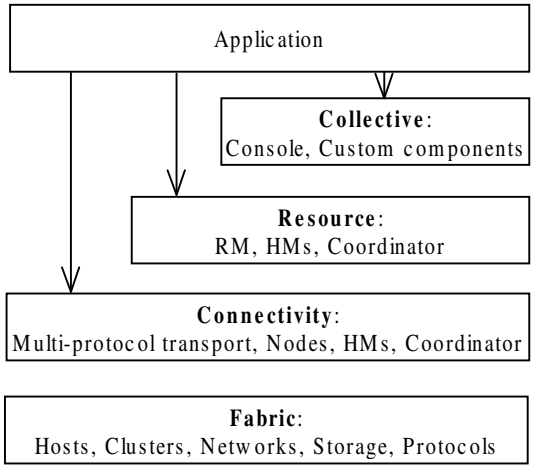
Our proposal (2)

- The middleware adopted is HiMM
  - *Hierarchical Metacomputer Middleware*
- The programming model adopted is the one provided by ProActive
  - Currently ProActive is implemented on top of RMI that does not offer specialized services for Grid computing
- This way
  - an application can be programmed by exploiting the asynchronous remote method invocation model
  - metasystems can be transparently managed by using meta-objects
    - through meta-objects, programmers can exploit the services of the underlying middleware without affecting the functional code of the application.

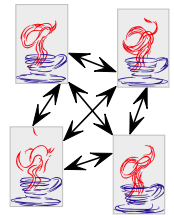


A brief introduction to HiMM (1)
layered architecture

- HiMM has been developed according to the layered architecture proposed by Foster et al.

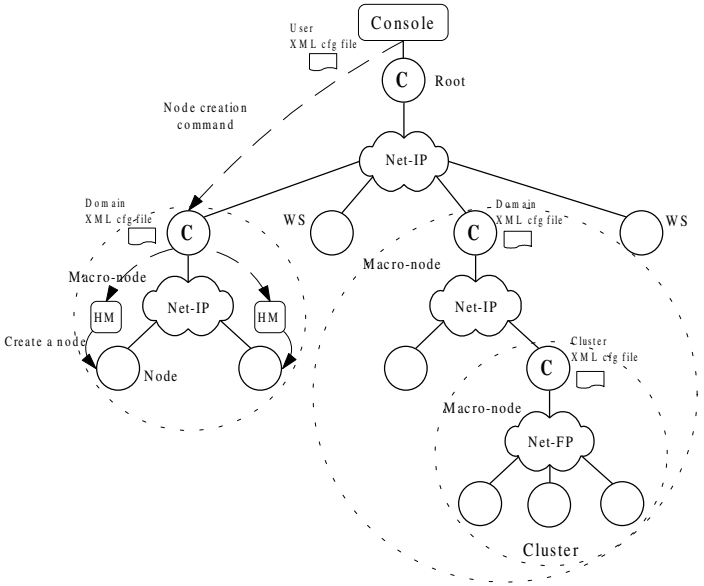


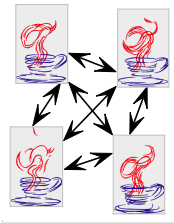
- Each level adds abstractions
- At fabric layer, it allows a user to exploit collections of hosts, which can be workstations or computing units of parallel systems, interconnected by heterogeneous networks



A brief introduction to HiMM (2)
connectivity layer

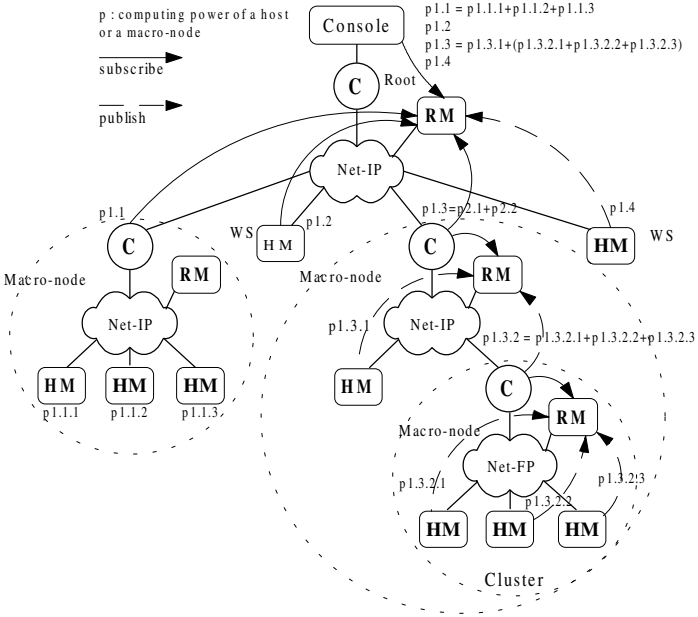
- HiMM manages resources according to a hierarchical topology (**Hierarchical Metacomputers**) in order to
  - meet the constraints of the Internet organization
  - exploit the heterogeneity of networks
  - improve scalability
- Clusters of computers hidden from the Internet or intra-connected by fast networks are seen as *macro-nodes*
  - A macro-node is a high-level concept that allows clusters to be transparently used as a single powerful machine
  - A macro-node can in turn contain other macro-nodes
    - A metacomputer can be organized according to a **recursive tree topology**
- The **Coordinator** interfaces the macro-node with the metacomputer network

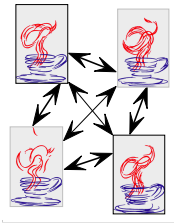




A brief introduction to HiMM (3)
resource layer

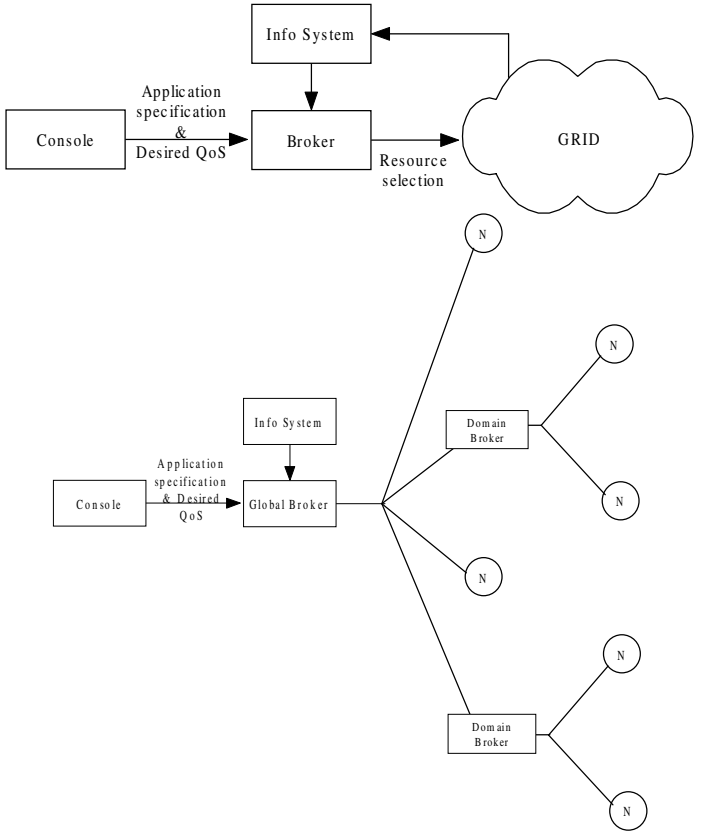
- A macro-node is characterized by two main components:
  - The **Host Manager (HM)**
  - The **Resource Manager (RM)**
- The HM runs on each node wanting to donate CPU cycles
- The RM is use to publish the available computing power at each level
- A macro-node manages another important component:
  - the **Distributed Class Storage System (DCSS)**
  - This component allows a HiMM to run applications even if application code is not present on nodes

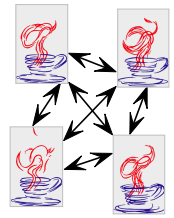




A brief introduction to HiMM (4)
collective layer

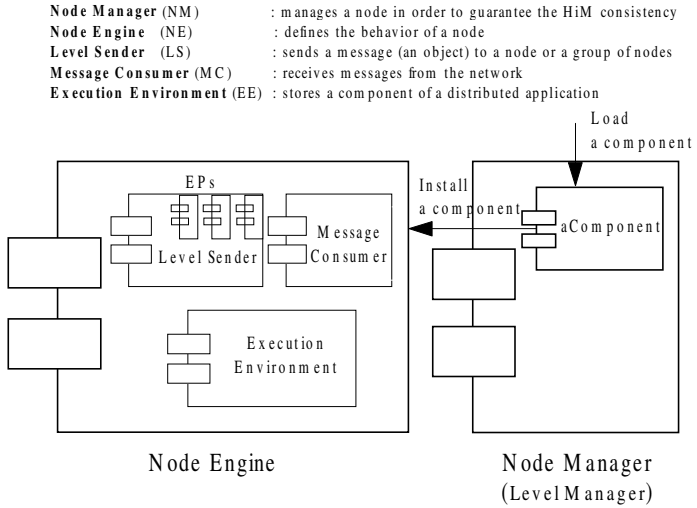
- The **broker** acts as a mediator between the user (console) and grid resources (HiM) by using middleware services (Info System and HMs)
  - the broker becomes responsible for resource discovery, resource selection, process/task mapping, task scheduling, and presents the Grid (HiM) as a single, unified resource
- The architecture of the broker is distributed and hierarchical in order to mach the architectural organization of a HiM

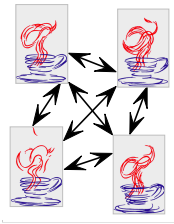




A brief introduction to HiMM (5)
node architecture

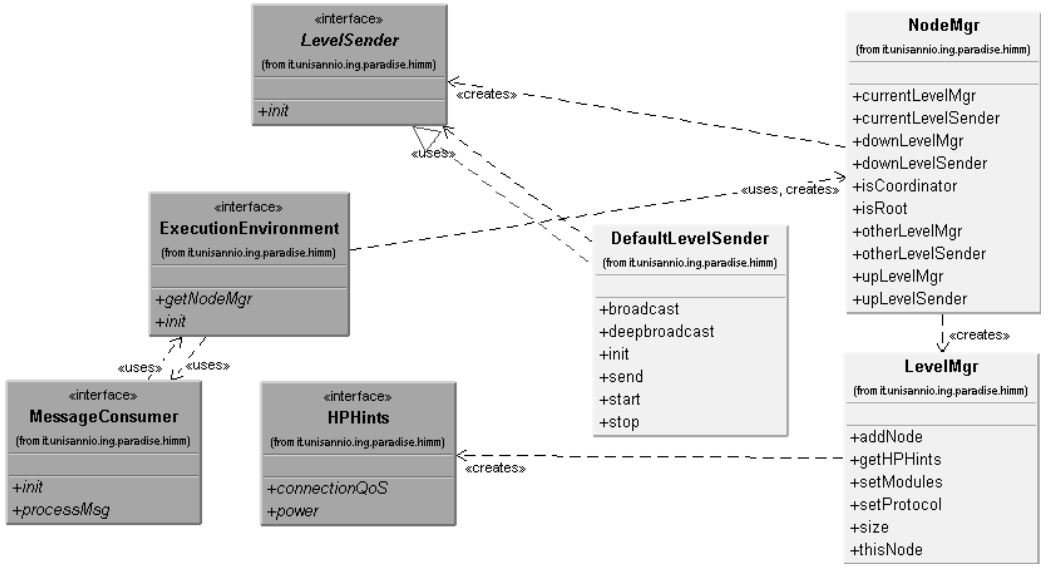
- HiMM implements its services in several software components whose interfaces are designed according to a **Component Framework** approach
- Both nodes and coordinators are processes in which a set of software components are loaded either at start-up or at run-time
- The main components are:
  - the **Node Manager (NM)**
    - guarantees macro-node consistency
    - provides users with services for writing distributed applications
    - takes charge of some system tasks, such as the creation of new nodes at run-time
  - the **Node Engine (NE)**
    - The **Message Consumer** (MC)
    - The **Execution Environment** (EE)
    - The **Level Sender** (LS)

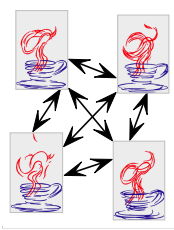




A brief introduction to HiMM (6)
API

- A component is a Java class implementing one or more interfaces according to the pattern “Inversion of control”

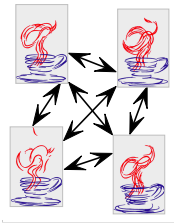




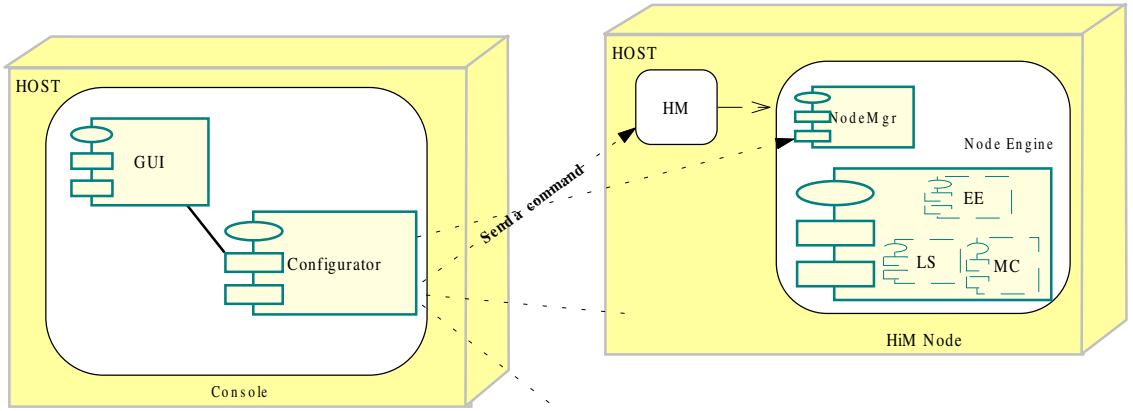
A brief introduction to HiMM (7)
communication API

- HiMM allows nodes to communicate using the component **`LevelSender`**
- A **`LevelSender`** can be customized even if a default one is always available
  - This component provides users with simple communication mechanisms based on the asynchronous sending of objects

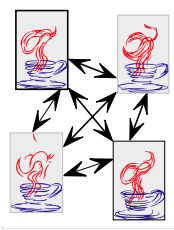
```
class DefaultLevelSender impements LevelSender {
  public void send(Object m, int node) { … }
  public void broadcast(Object m) { … }
  public void deepBroadcast (Object m) { … }
}
```



A brief introduction to HiMM (8)
interaction among components

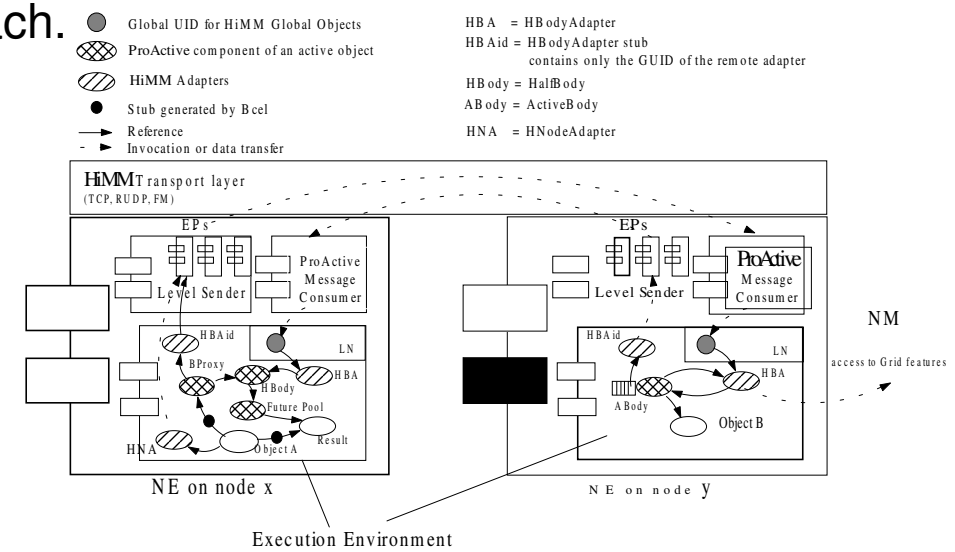


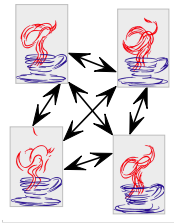

```
class Application implements ExecutionEnvironment {
  void init(NodeMgr nm) throws … {
    nodeMgr = nm;
  }
  void start() {
    …
    nodeMgr.downLevelSender().send(msg, node);
    nodeMgr.downLevelMgr().addNode(…);
    Object o = nodeMgr.downLevelMsgConsumer().receive(); …
  }}
```



Integration HiMM / ProActive (1)

- The implementation of P/H has been made easy thanks to the particular organization of both software systems
  - ProActive is heavily based on the Adapter Pattern
  - HiMM is implemented following the Component Framework approach.

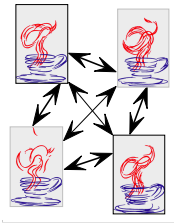




Integration HiMM / ProActive (2)

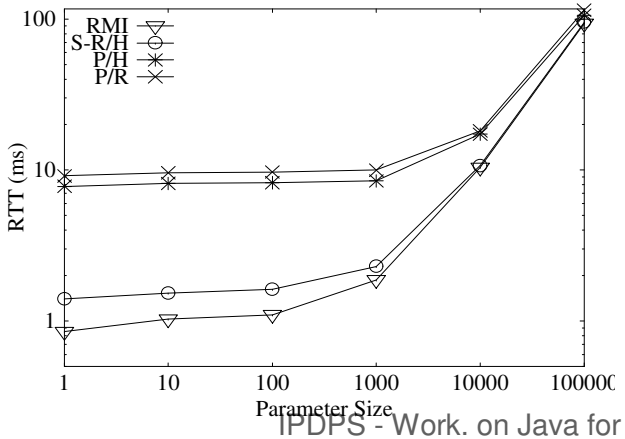
```
public class MatrixMultiply extends
                    ProActiveExecutionEnvironment {
  private NodeMgr nodeMgr;
  private volatile boolean stop = false;
  public void init (NodeMgr nm) {
    super.init(nm); nodeMgr = nm;
    NodeFactory.setFactory("himm", new HNodeFactory());
  }
  public void start() {
    stop = false;
    if(nodeMgr.isRoot()) {
      HMatrix[] mDxActiveGroup=null; int dim=0;
      System.out.println("Insert matrix size:");
      <read dim>;
      Matrix mDx = new Matrix(dim);
      LevelMgr lm = nodeMgr.downLevelMgr();
      int size = lm.size()-1;
      Object[] po = new Object[]{mDx.getTab()};
      mDxActiveGroup = new Matrix[size];
      for(int i =0; i<size; i++)
      mDxActiveGroup[i] = (HMatrix)
          ProActive.newActive(HMatrix.class.getName(),
          po, NodeFactory.getNode("himm://"+(i+1)), null,
          HMetaObjectFactory.newInstance());
      Matrix mSx = new Matrix(dim);
      while (!stop) {
        Matrix[] results = multiply(mSx, mDxActiveGroup);
        Matrix result = Matrix.rebuild(results, dim);
        ...
      }
    }
  }
…
}
```

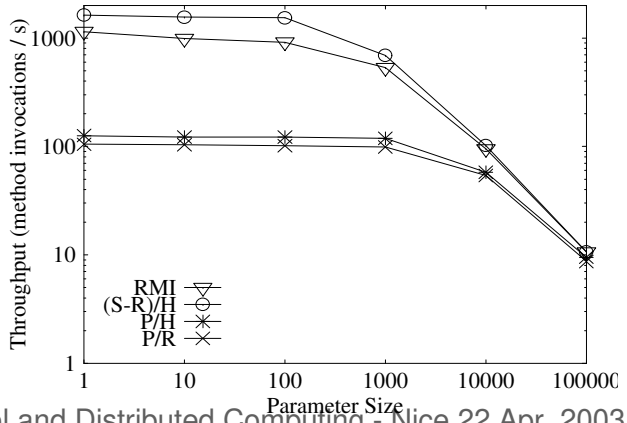
```
public class HMatrix extends Matrix implements InitActive, RunActive, EndActive {
  private HMatrix[] subMats;
  public void initActivity(Body body){
    HBodyAdapter hba =(HBodyAdapter) body.getRemoteAdapter();
    NodeMgr nodeMgr = hba.getNodeMgr();
    if(nodeMgr.isCoordinator()) {
      LevelMgr lm = nodeMgr.downLevelMgr();
      int size = lm.size()-1; subMats = new Matrix[size];
      Object[] po = {tab};
      for(int i =0; i<size; i++) {
        subMats[i] = (HMatrix)ProActive.newActive(HMatrix.class.getName(), po,
            NodeFactory.getNode("himm://"+(i+1)),null,
            HMetaObjectFactory.newInstance());
      }
    }
  }
  public void runActivity(Body body) {
    Service service = new Service(body);
    while (body.isActive()) {
      Request r = service.blockingRemoveOldest();
      HBodyAdapter hba =(HBodyAdapter) body.getRemoteAdapter();
      NodeMgr nodeMgr = hba.getNodeMgr();
      if(r.getMethodName().equals("multiply")&& nodeMgr.isCoordinator()){
        HiMMRequest rr = (HiMMRequest)r;
        float[][]mSxtab =(float[][])rr.methodCall.getParameter(0);
        Matrix mSx = new Matrix(mSxtab);
        Matrix[] results = new Matrix[subMats.length];
        Object[] subSxMats =  mSx.createSubMatrixes(subMats.length);
        for(int i=0; i< subMats.length; i++)
          results[i] = subMats[i].multiply((float[][])subSxMats[i]);
        Matrix result = Matrix.rebuild(results, mSxtab.length);
        Object[] params = new Object[]{result};
        Class[] classes = new Class[]{Matrix.class};
        try {
          Method m = HMatrix.class.getMethod("getResult", classes);
          rr.methodCall = MethodCall.getMethodCall(m, params);
        } catch  (NoSuchMethodException e){ … }
        service.serve(rr);
      } else service.serve(r);
    }
  }
}
```

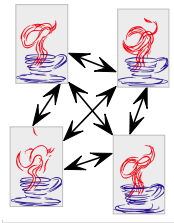


Performance analisys (1)

- We have conducted a first analysis to compare RMI with HiMM and P/R with P/H
- The benchmark is the invocation of a remote method (with one parameter and an empty body) and the return of an integer value, for a varying size of the parameter
- The figure shows that for a small size (1 byte) of the parameter, RMI RTT (0.9 ms) is smaller than HiMM RTT (1.3 ms), but P/H RTT (7.8 ms) is smaller than P/R RTT (9.2 ms)
  - Even if the HiMM transport layer is not optimized, P/H behaves better than P/R due to the use of asynchronous messaging that allows low-level ProActive operations to be overlapped with communication

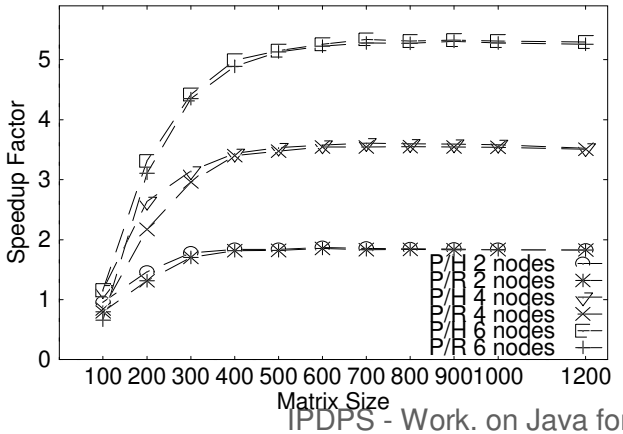


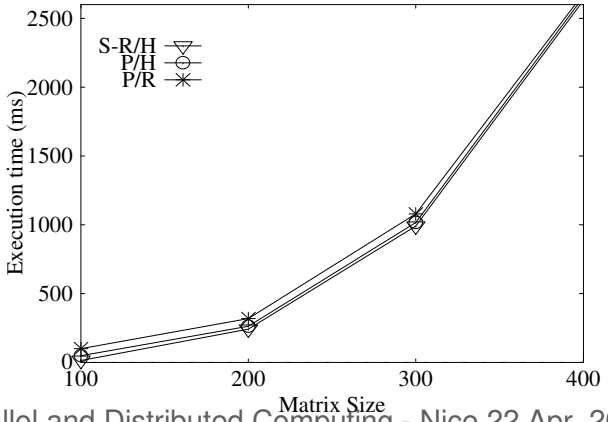


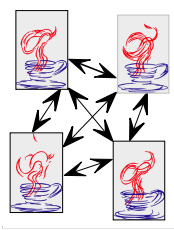


Performance analisys (2)

- A further analysis has aimed to measure the speedup factor by running a simple application benchmark
- The benchmark is the product of two square matrixes with different sizes of the matrixes.
- The performance obtained with P/H is slightly better than the one obtained with P/R, especially when the size of the matrixes is small
  - This is mainly due to the improvement of remote method invocation implemented by HiMM
  - When the size of the matrixes is large, the execution time is dominated by the time of the matrix serialization, which is the same in P/H and P/R

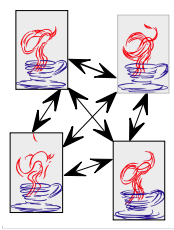






Conclusions

- I have described the integration of ProActive with HiMM to create a new framework for easily programming Grid applications with the active object model
- The separation of concerns and the use of meta-objects allow programmers to exploit the underlying features of HiMM to program distributed aspects of applications without affecting functional code
  - This approach enables code reuse and makes the object oriented approach effective for the development of distributed and parallel applications
- HiMM assures better performance than RMI to ProActive
  - The overhead introduced by the new framework is small



Future work

- In the future, we will conduct further experiments with a larger, hierarchical distributed system
  - characterized by different computational power, communication hardware and protocols
- We will complete the definition and the implementation of the broker architecture
- We will define a framework for automatically transforming sequential object oriented applications in parallel and distributed applications when the master/slave parallel model can be adopted