

Jacob : a software framework to support the development of e-services, and its comparison to Enterprise JavaBeans

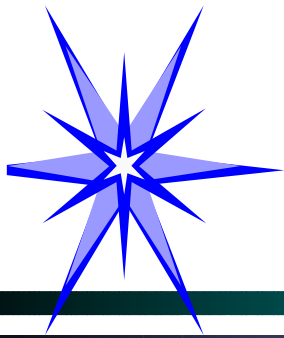


Serge Chaumette and Pierre Vignéras

Distributed Systems and Objects team

LaBRI, Université Bordeaux 1, France

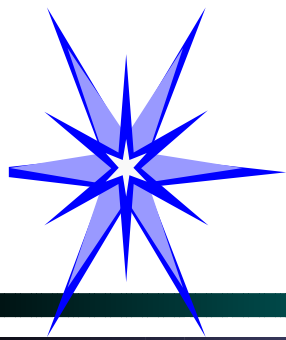
{chaumett, vigneras}@labri.fr



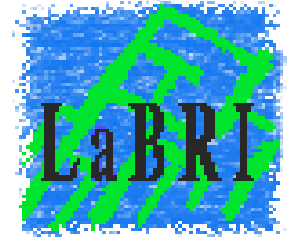
Aim of the talk



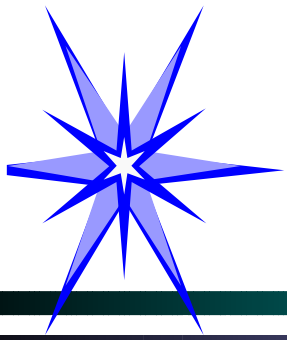
- Describe the EJB™ framework
 - ◆ Expose the related problems
- Propose the Jacob framework
 - ◆ solutions to the above problems
 - ◆ new problems Jacob bring out
 - ◆ solutions to those problems
 - ◆ Keywords
 - Java™, EJB™, RMI™, Active containers, Distributed Systems, Components, middleware, multitier



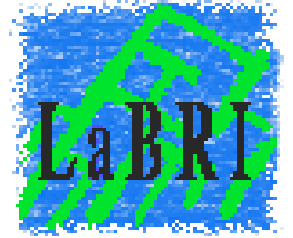
Outline



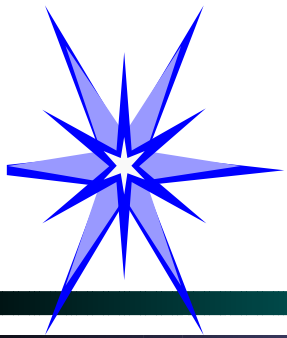
- The multitier concept
- Overview of the EJB Framework
- Problems related to EJBs
- Solutions proposed by Jacob
- The Jacob concept : active containers
- The Jacob framework
- Problems related to Jacob and solutions
- Conclusion and future work



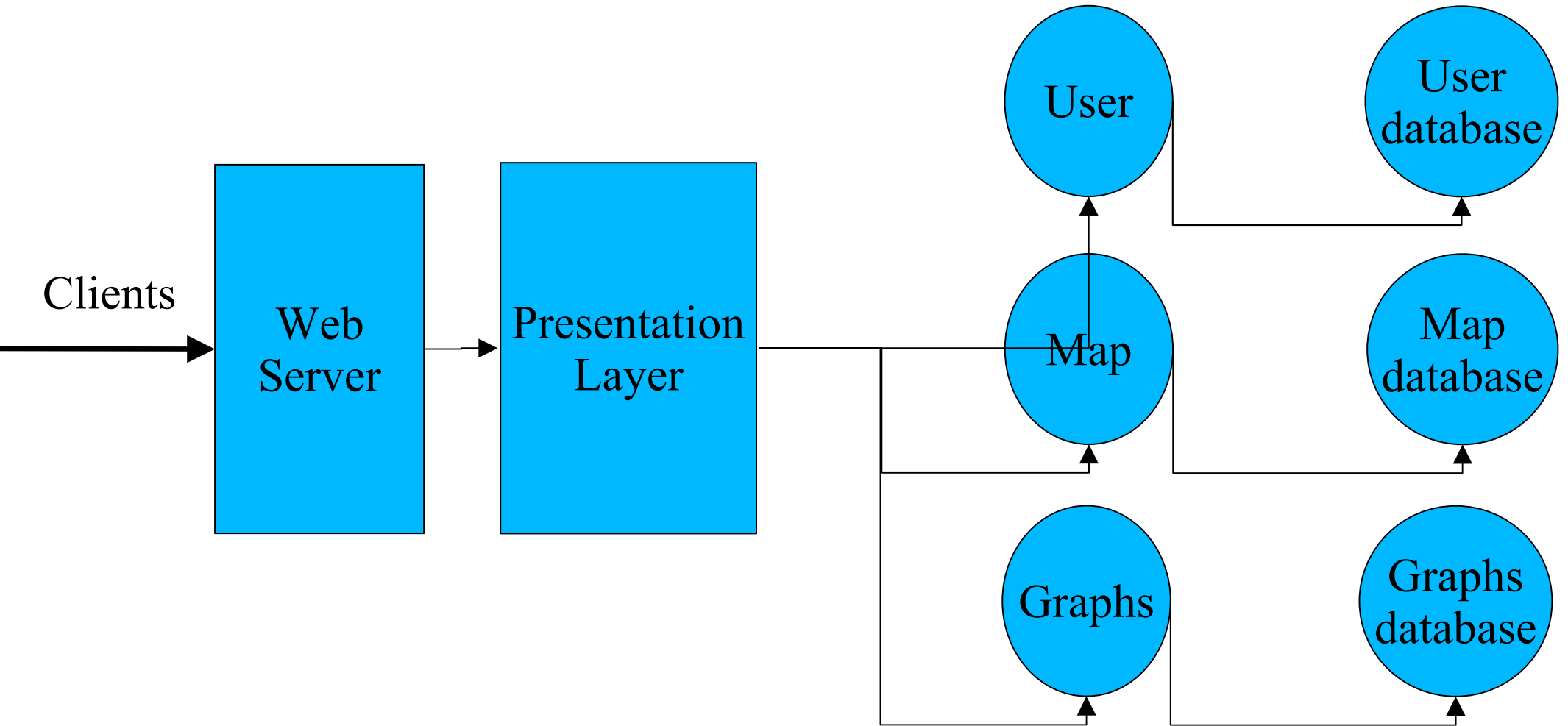
The multitier concept

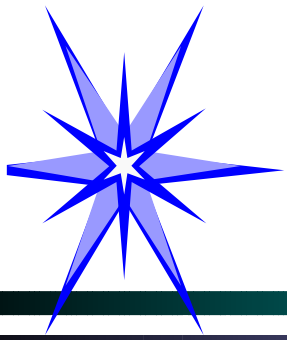


- “Old” client/server approach is problematic
 - ◆ client is tied to its server
 - high maintenance cost
 - ◆ low disponibility
 - ◆ low scalability
- The multitier approach
 - ◆ separate applications in independent distributed layers
 - ◆ high disponibility
 - ◆ high scalability

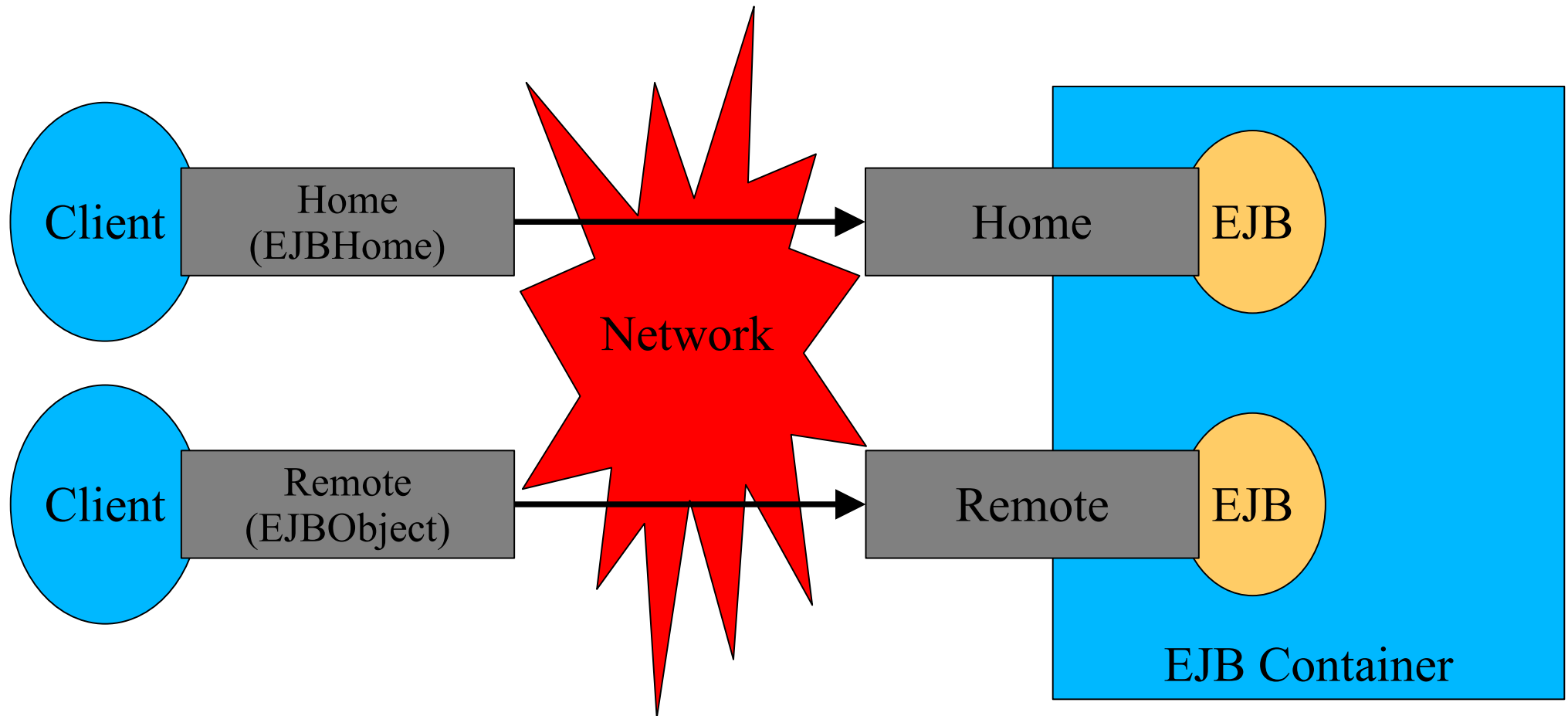
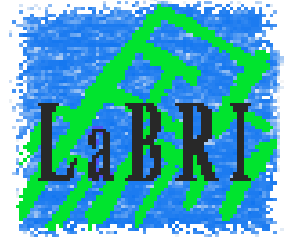


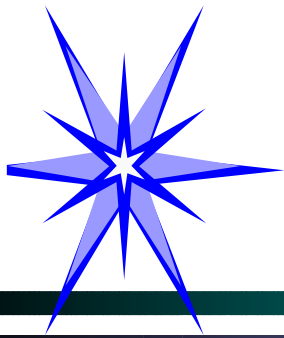
Example : e-map server





Overview of the EJB framework

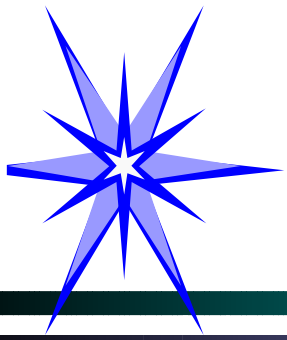




Overview of the EJB framework



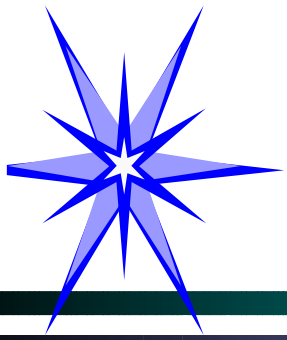
- An EJB
 - ◆ lives in an EJB container which provides services
 - persistency, security, versioning, transaction, etc.
 - ◆ is a component with two interfaces
 - Home interface
 - Remote interface (business methods)
- EJB clients
 - ◆ manipulate EJB with the Home interface
 - ◆ communicate with EJB with the Remote interface



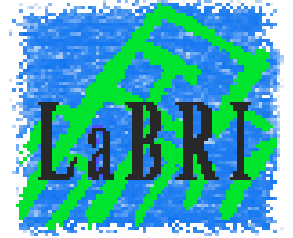
Problems related to EJBs



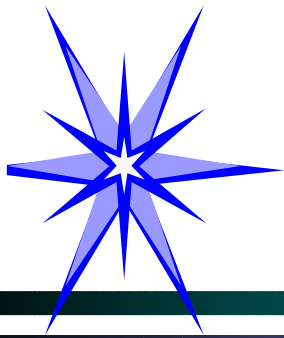
- Statically distributed system
 - ◆ Home and Remote interfaces
 - ◆ Specific inheritance is needed (EntityBean or SessionBean)
- False strong typing
 - ◆ create method (Home interface) require an ejbCreate method (Bean implementation)



Problems related to EJBs



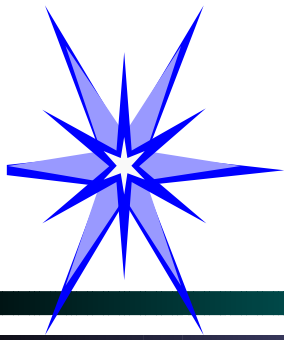
- Multithreading and re-entrance is problematic
 - ◆ Loopback code is strongly discouraged
 - ◆ Coarse object re-entrance vs fine method re-entrance
- The code must conform to the EJB specification programming restrictions (23.1.2 p 462 v2.0)
- EJB components are not reusable in another context



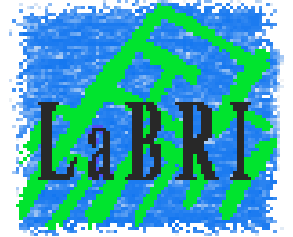
Solutions proposed by Jacob



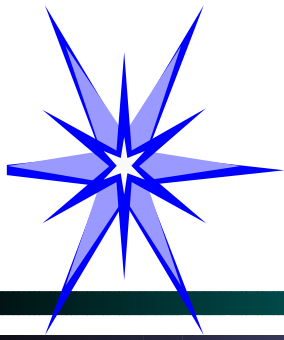
- Focus on dynamicity
- Any object can become remote at runtime
 - ◆ Neither interface declaration nor source compilation => legacy code
 - ◆ Remote aspect is important for clients
- Any object method may be called asynchronously
 - ◆ Distribution and asynchronism => parallelism
 - ◆ asynchronism and distribution => computation/communication overlapping



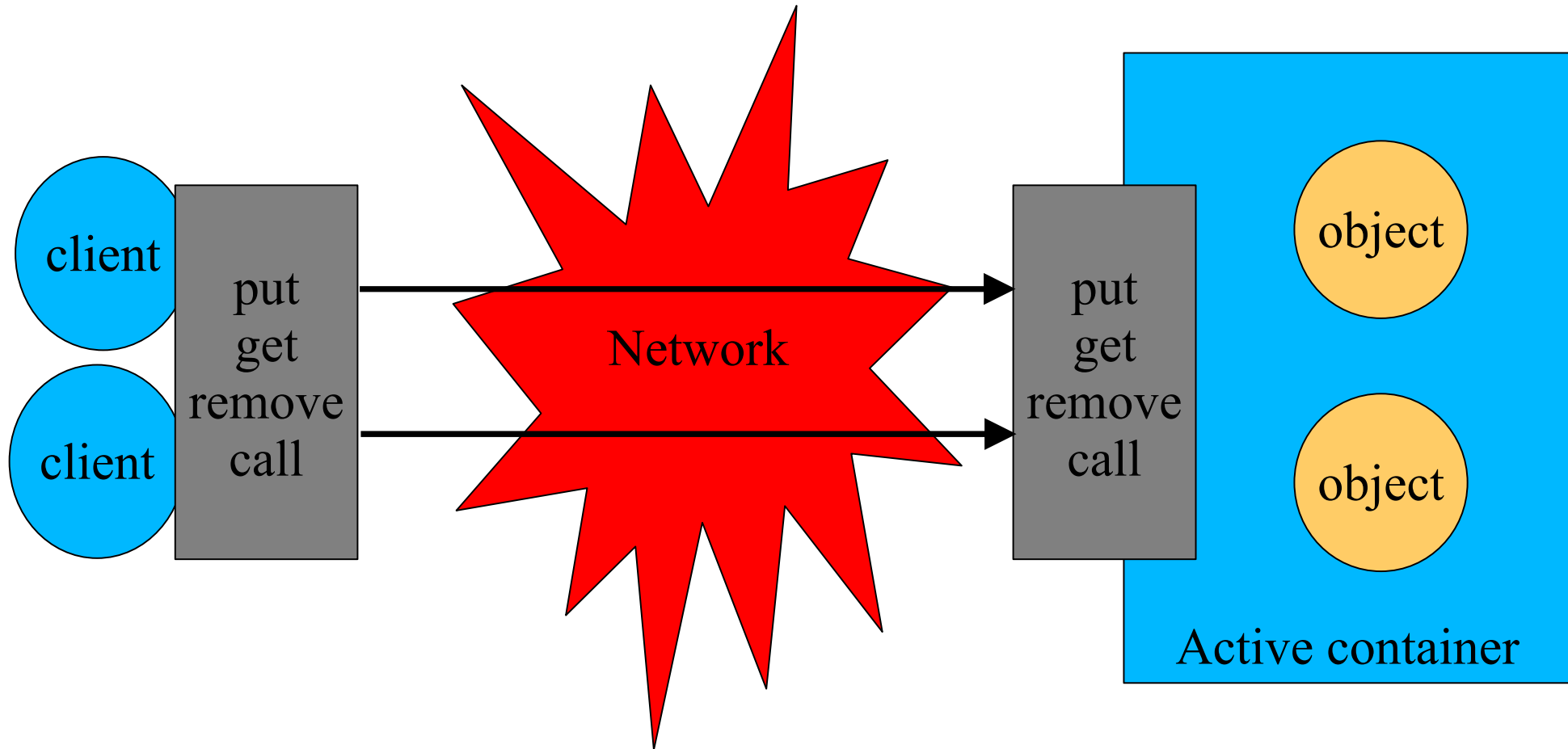
The Jacob concept : *active containers*

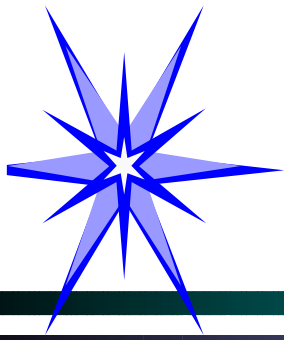


- A container
- Four methods
 - ◆ Object put(Object key, Object object)
 - ◆ Object remove(Object key)
 - ◆ Object get(Object key)
 - ◆ void call(Object key,
String method,
Object[] args,
MethodResult result)
- The call method generates activity



The Jacob concept : *active containers*

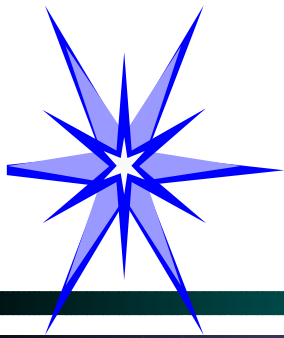




The Jacob concept : *active containers*



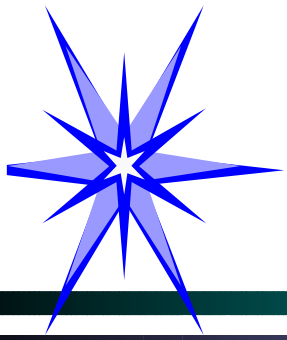
- Manipulation of Jacob objects with `put`, `get` and `remove`
- Communication with Jacob objects with `call`
- Client's Jacob object reference
 - ◆ a pair (*activeContainer*, *key*)
 - *light object* => *Efficient serialization*
 - ◆ Easy extension to distributed objects
 - n-pair
- *key is any object*
 - ◆ *Some services may use this property*
 - *Security, transaction, persistency, ...*



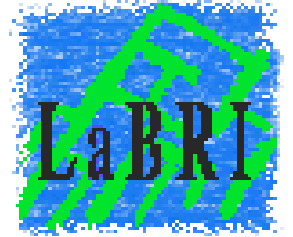
The Jacob concept : *active containers*



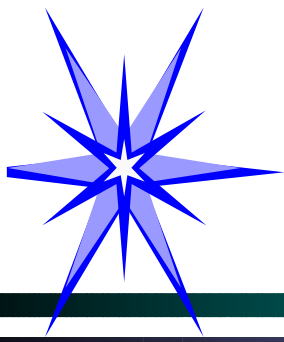
- Active containers have been modeled in pi-calculus
- Active containers are expressive : `agent.migrate(destination)`
 - ◆ `Agent agent = (Agent) from.get(agentID);`
 - ◆ `from.remove(agentID);`
 - ◆ `to.put(agentID, agent);`
 - ◆ `to.call(agentID, "onMigrationMethod", null, null);`
- Services are classical objects in Jacob
 - ◆ Service on demand by object
 - Locality, persistency, security, versioning, ...
 - ◆ Active containers are “light objects”
 - Easy code maintenance, low resources consumers (important for embedded systems for example)



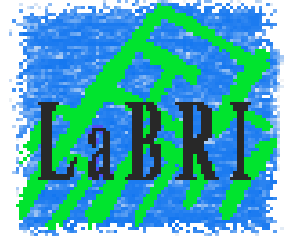
Server-side asynchronism



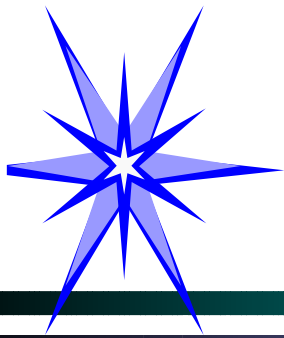
- void call(Object key,
String method,
Object[] args,
MethodResult result)
- The specified method is executed by a new thread of the container
 - ◆ This thread is remotely accessible to the client
- The result (exception or return value) is sent to the remote object MethodResult
- *Server-side asynchronism*



Asynchronous remote method invocation



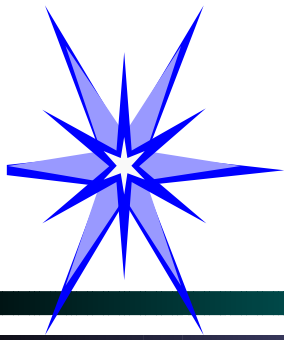
- The previous call requires a network communication
 - ◆ the call is blocked until the server-side thread starts
 - ◆ *partial asynchronism*
- Clients can use a *client-side asynchronism library* for every method of the active container (put, get, remove and call).
- Clients can access both threads implied in a client-side asynchronous call method invocation :
 - ◆ the client-side thread that calls the server-side asynchronous call method
 - ◆ the server-side thread that runs the specified method



Asynchronism and exceptions handling



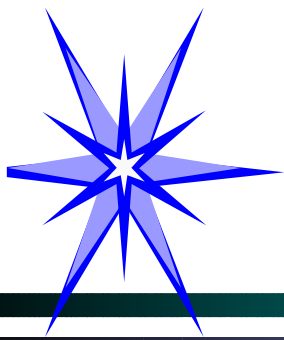
- Two types of exceptions
 - ◆ network related exceptions
 - handled in a generic manner
 - managed by ExceptionHandler
 - ◆ “Business” related exceptions
 - handled in a per call basis as usual on *result recovering*
 - handled as soon as desired by the “*earlier warning*” mechanism
 - `callerThread.interrupt()`



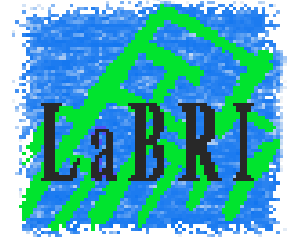
Problems related to Jacob and solutions



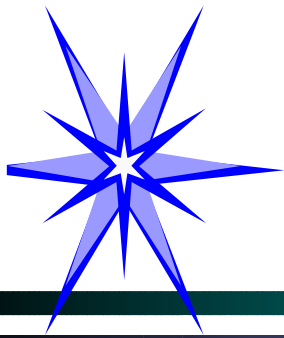
- Massive use of reflection is slow
 - ◆ No solution since it provides dynamicity
- Remote objects (ActiveContainer, MethodResult, RemoteThread) are heavy RMI objects
 - ◆ Implement lighter remote objects
 - Efficiency
 - ◆ Implement more configurable protocol
 - use more specific hardware (myrinet, SCI, etc.)
- Strong typing is lost since a string specifies the method to run in call method
 - ◆ Compilation of specific client interface
 - ◆ Use of the new JDK 1.3 Proxy



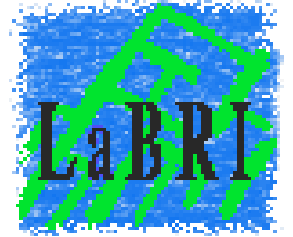
EJB vs Jacob



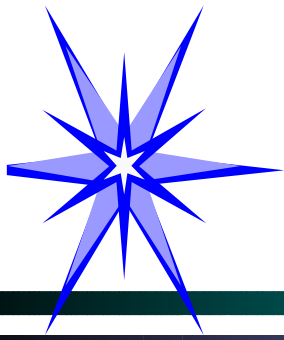
- EJB Clients need two distinct interfaces : Home for manipulation and Remote for communication
- Remote objects are RMI objects
- Client EJB reference is a heavy weight object (EJBObject + RMISub)
- Every call is synchronous
- Every call may throw a RemoteException
- Object manipulation and communication are provided by the single active container's interface
- Any object can become remote at any time
- Client remote object reference is a light weight object (activeContainer, key)
- Full asynchronism
- Two different exception handling mechanisms (network related and method related)



EJB vs Jacob



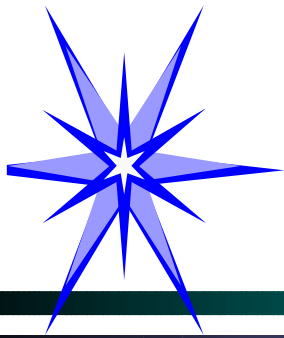
- Interface declaration and compilation
- Specific inheritance
- Strong typing (partial)
- Coarse object re-entrance
- Programming restrictions
- EJB are not reusable in another context
- No declaration or compilation required
- Inheritance not required
- Strong typing (interface)
- Fine method re-entrance
- No restrictions
- Object has not to be designed to be in an active container



Conclusion and future work



- **Jacob is an operational framework**
 - Calculation of Pi
 - The traveling salesman problem
 - Distributed spectral analysis of wave sound
 - Generic load balancer
 - Implementation of a more efficient protocol for remote objects
 - Automatic distribution and parallelisation of Java programs
 - MASIF implementation
- **Services objects are needed :**
 - persistency, security, transaction, etc.

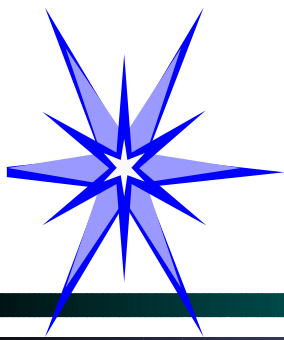


Examples

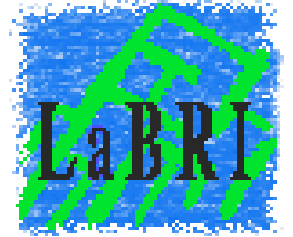


▪ Deployment : multiple asynchronous insertions

- Object[] array = ... // Objects to deploy
- AACProxy[] proxy = ... // active containers
- for(int i = 0; i < proxy.length; i++) {
 PutTask putTask = new PutTask(proxy[i], o);
 putTask.setTerminatedTaskHandler(new TerminatedTaskHandler() {
 public void done(Task task) {
 callTask[i] = proxy.call(putTask.getKey(),
 method[i],
 args[i],
 result[i]);
 }
 });
 putTask.add(); // insert the task in the thread pool
}

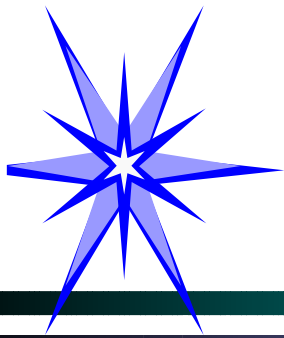


Examples



▪ Asynchronous insertion

- Object o = new MyObject();
- AACProxy proxy = AACProxy.getInstance(activeContainer);
- proxy.setExceptionHandler(new ExceptionHandler() {
 public void handleException(Task task) {
 task.getThrowable().printStackTrace();
 // Exception handling here
 });
- PutTask putTask = proxy.put(o); // the key is automatically generated
- computeSomething();
- putTask.waitUntilDone();



Examples



- “*Earlier warning*” exception handling mechanism
 - `MethodResult result = proxy.call(key, method, args).getResult();`
 - `while(!result.isResultAvailable()){`
 - `doSomeLittleWork();`
 - `if (Thread.isInterrupted() && result.isAvailable()) break;`
 - `else { // someone else interrupted us }`
 - `endWork();`
 - `}`
 - `try{`
 - `MyResult myResult = (MyResult) result.getReturnedResult();`
 - `}catch(Throwable t){ // Exception thrown by the remote method`
 - `handleException(t);`
 - `}`