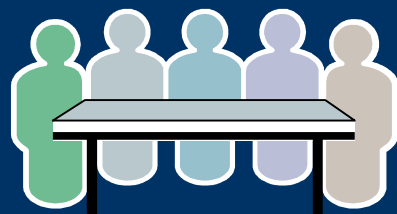
Final Year Project Presentation

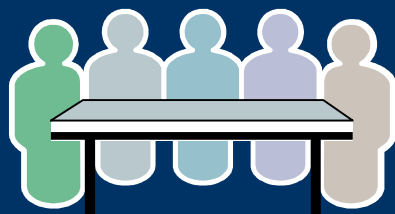
A framework for an agent-based development environment with jini/javaspace(Real time collaboration)

- Supervisor: Dr. Stephen Chan
- Co-examiner: Dr. Jane Wong
- Student: Yim Wai Hin



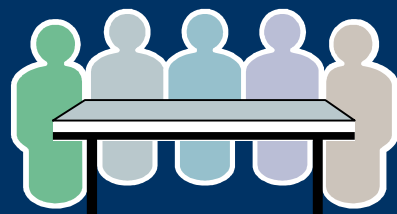
Motivation

- The IDE available on the market are not easy to extend the function.
- Location limitation - the user need to install and maintain same IDE for many machines in the office.
- Lack of real time collaboration support.



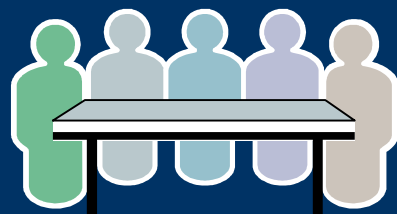
Objective

- Build a distributed development environment. Users can dynamically add and remove tools
- Allow multiple users to access / edit the same model
- Using multiple version concurrency control (MVCC) in place of locking to resolve conflicting actions in the collaborative system



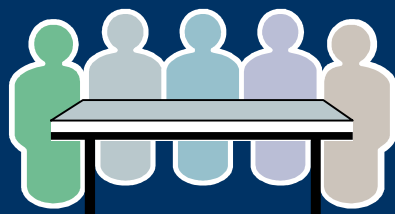
MVCC Vs. Locking

- Locking
  - Avoid conflicts by ensuring that only one user can access an object at any time
  - Poor performance of concurrency is a side effect
- MVCC
  - Less stringent, but more complicated

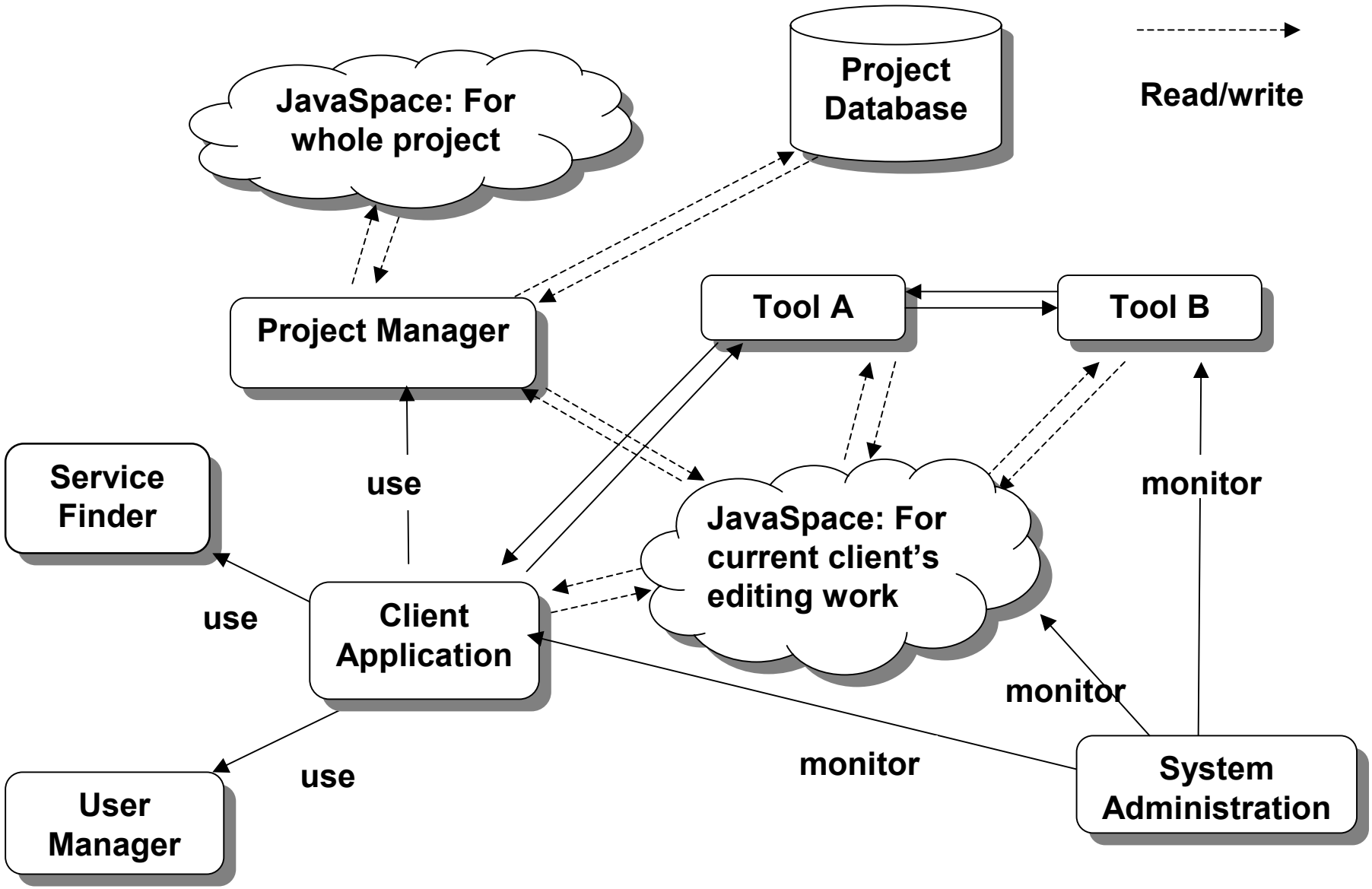


Methodology

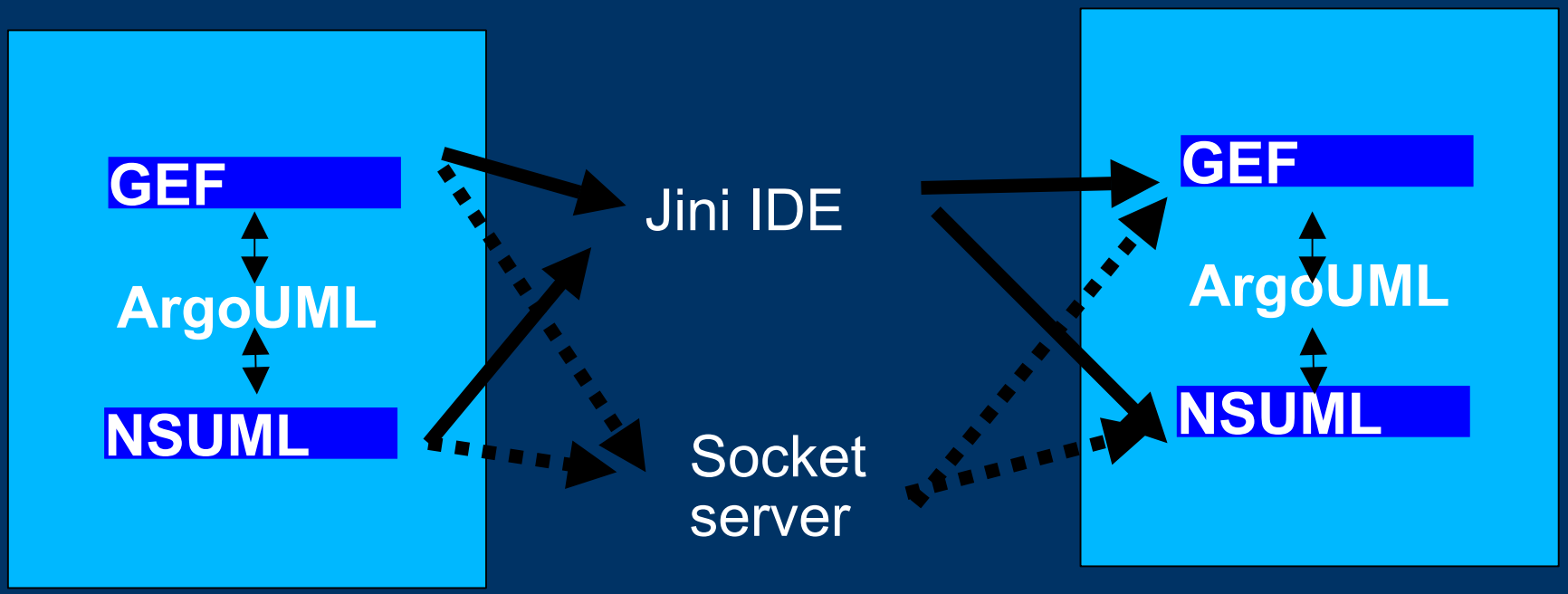
- Build on the jini development environment
  - Benefits of jini/javaspace
- Based on an opensource UML editor ArgoUML
  - The pros and cons of this UML editor
  - The pros and cons of using XMI file format, the de facto XML format of UML



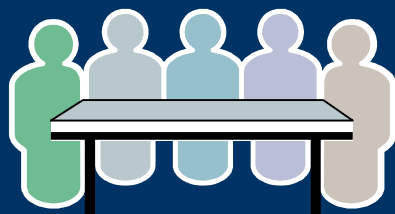
Architecture of Jini infrastructure



Architecture Overview

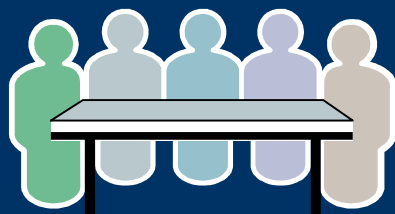


- GEF is the diagram drawing framework
- The system extract nodes and edges of the UML diagrams
- NSUML is the UML meta model framework
- The system extract the UML detail elements

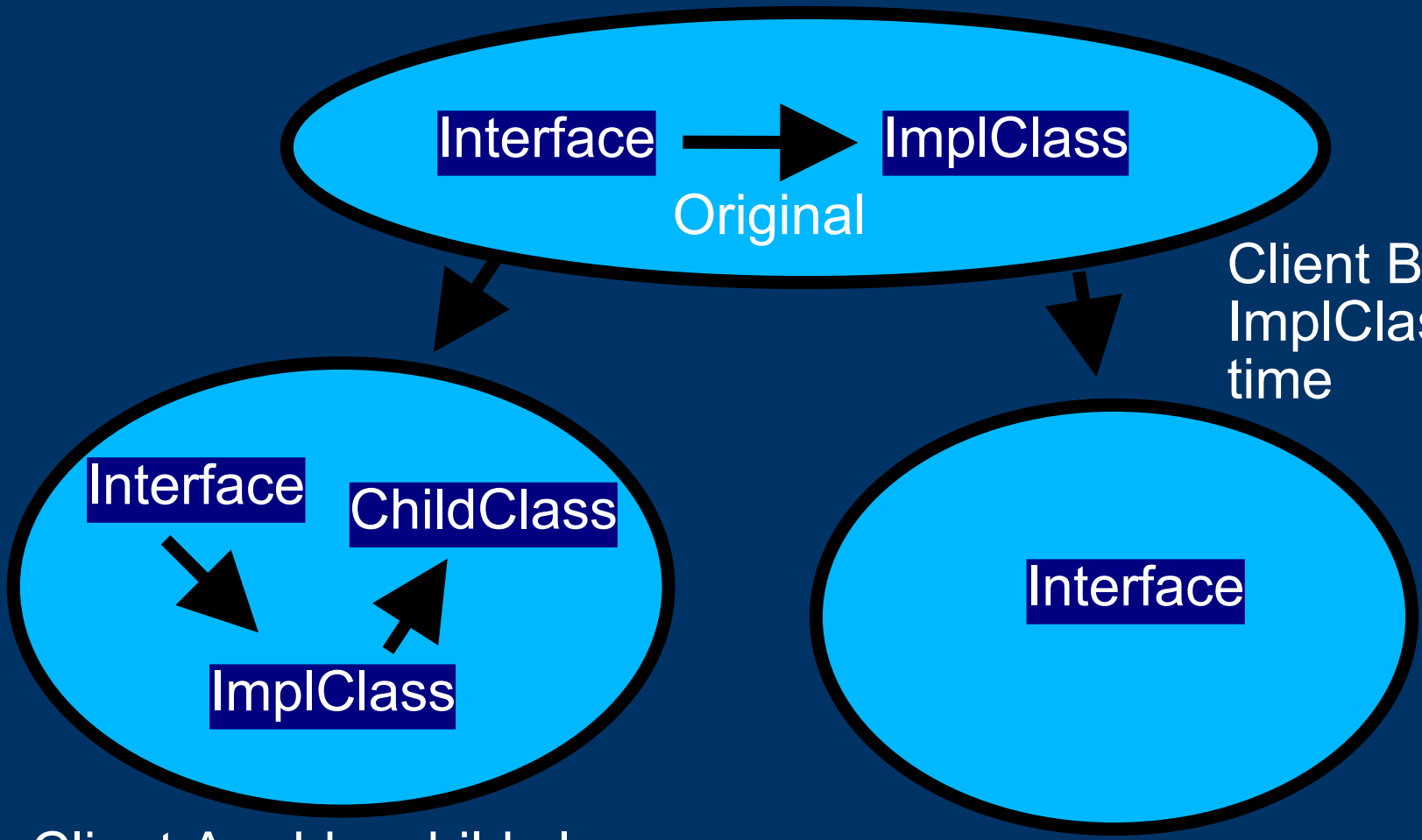


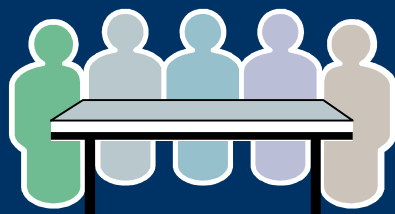
Proposed Algorithm

- UML diagram is a graphic.
- Every elements depend on some other elements, like attribute/operation depend on the model, child class depends on parent class.
- Check if the user deletes an element that have other elements depend on.



Example of How Algorithm Work

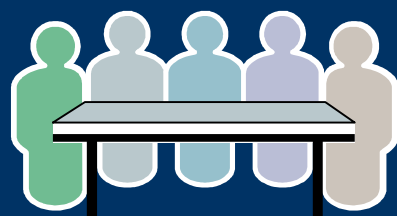




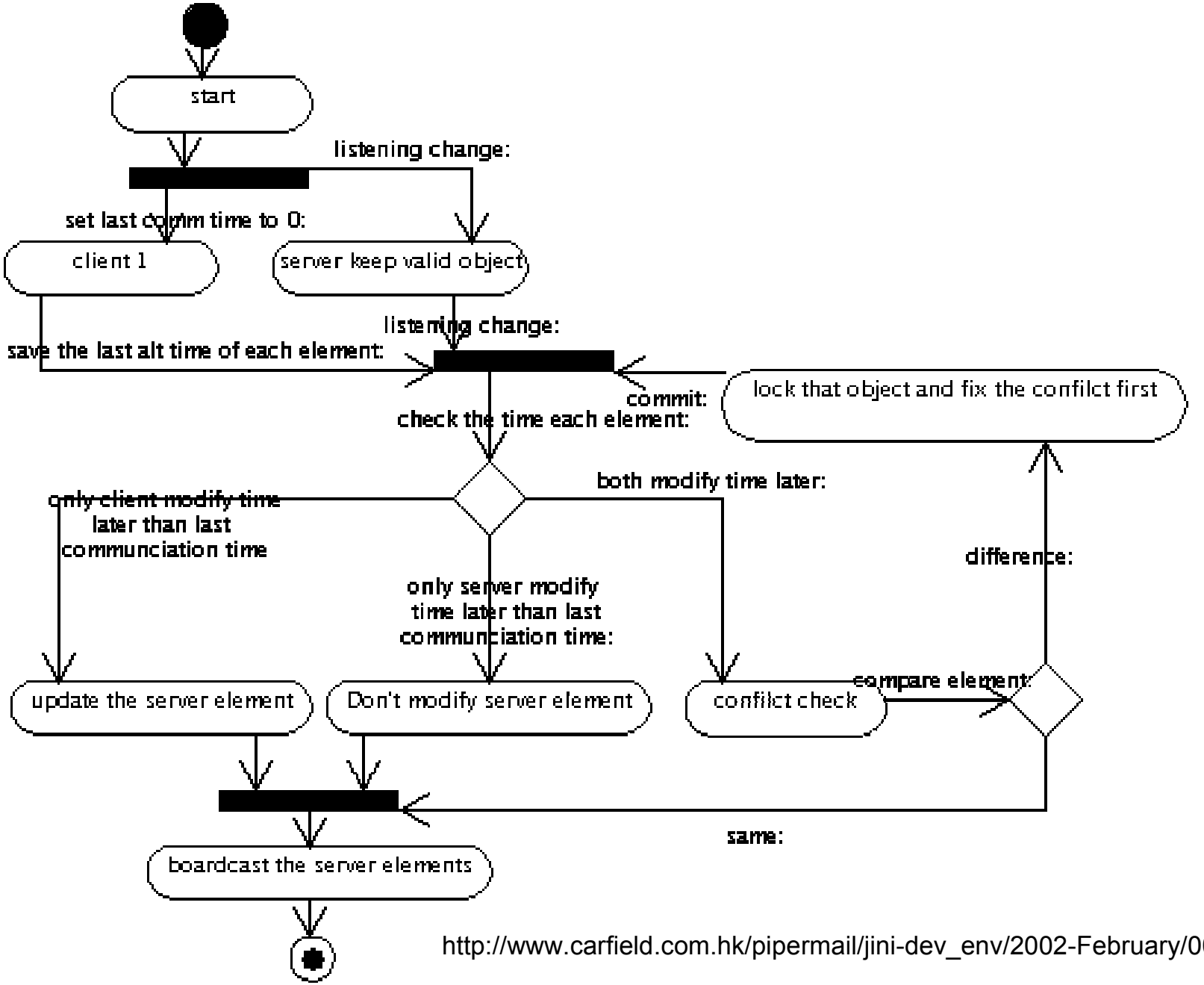
Problem and New Approach

- However after doing some prototype we find that this algorithm have some problems.
- It don't prevent all possible conflicts, e.g.: 2 clients modify same element in same time.
- Some work for the users will lost, because we will reject some change from the users.
- Borrow the idea from CVS[1] and propose other algorithm to solve the problem.

1: CVS: concurrency version system, more detail at http://www.cvshome.org.



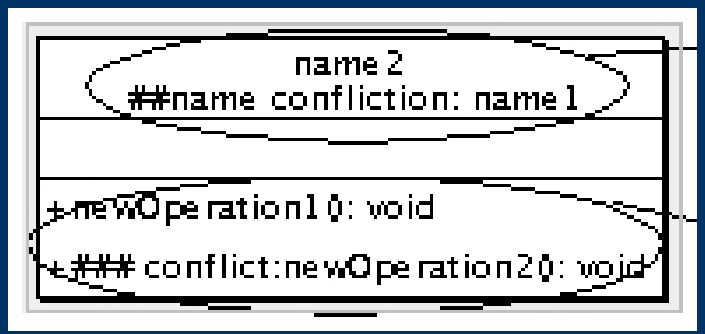
Conflict Resolution Flow

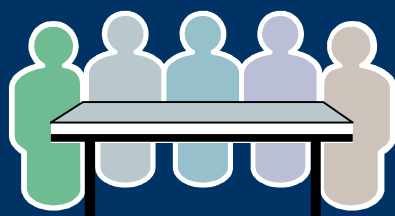


http://www.carfield.com.hk/pipermail/jini-dev_env/2002-February/000055.html

Conflict Resolution Model

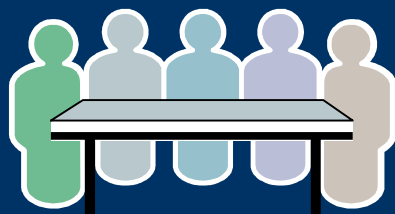
- There is no single basic element in UML.
- Due to the time limitation only concentrate on class diagram.
- Basic elements of class diagram including: name, stereo type,visibility, attributes and operations.
- Example of conflict model.





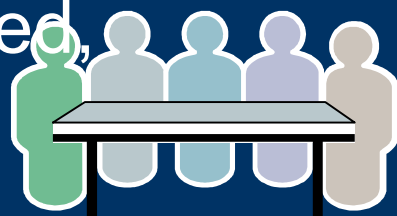
Result

- The patched UML editor successes to share the diagram between users
- It is able to detect the occurrence of conflict
- However, due to the time limitation, not much testing can be done, and the integration with jini is not complete



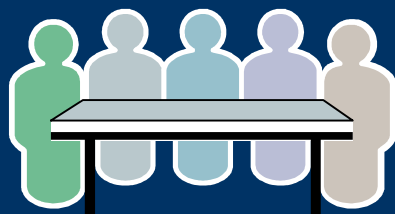
Conclusion of Using jini

- Able to solve some exceptions that is difficult to solve in traditional client-server model
- e.g.: In client server model, the versioning engine need to be called by many clients. It needs to be synchronized to keep the data valid; But once synchronized, there are possible to have dead-lock
- In jini/javaspace, the versioning engine is just a service in jini, which pick up the data at javaspace to process, no need to synchronized, no deadlock possible



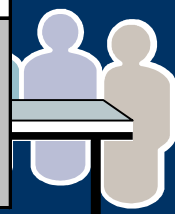
Conclusion of Using XMI

- Too complicated to process, many attributes we don't need.
- In order to use it, we need to spend many time to learn the specification. Which are not useful.
- It will be better to build our custom file format that only contain attributes we need.



Conclusion of Versioning Algorithm

	Locking	Versioning
User Experience	Poor, need to unlock everytimes after editing and need to wait for other unlock	Good, no need to wait, just resolve conflict occasional
System performance	Best	Good, a little CPU power spend on continue merging model, but it don't affect the user operation.
Network overheading	Very Low, only lock and unlock message	Low, need to transfer diagrams many times, but the the size are very small.
Implementation	Easy	Median, the algorithm are not complicate, but tedious to implement.



Future Work

- ArgoUML is not design for collaborative from ground up. We should replace ArgoUML with our implementation
- Improvement of the collaboration architecture - user control, versioning history management, WebDav integration
- Improvement of versioning algorithm – lock smaller attribute but not whole model, offline merging