

Final Year Project Presentation

A Framework for an Agent-based Development Environment with Jini/Javaspaces –

Internet Integrated Development Environment Framework (Internet-IDEF)



Supervisor: Dr. Stephen Chan Chi-Fai
Co-examiner: Dr. Korris Chung Fu-Lai
Student Name: Lam Hoi Kit
Student ID: 98247632D



Agenda

- Why? What? How?
- System Architecture
- Brief Jini Concept & JavaSpaces Concept
- Design and Implementation
- Validation – Remote Java Compiler and Collaborative UML Editor
- Challenges
- Conclusion



Why? What? How?

- Why?
 - Software development projects require variety of tools to accomplish tasks
 - IDE does not support particular tools required by specific projects
 - Location limitation
 - Low extensibility of existing IDE



Why? What? How? (con't)

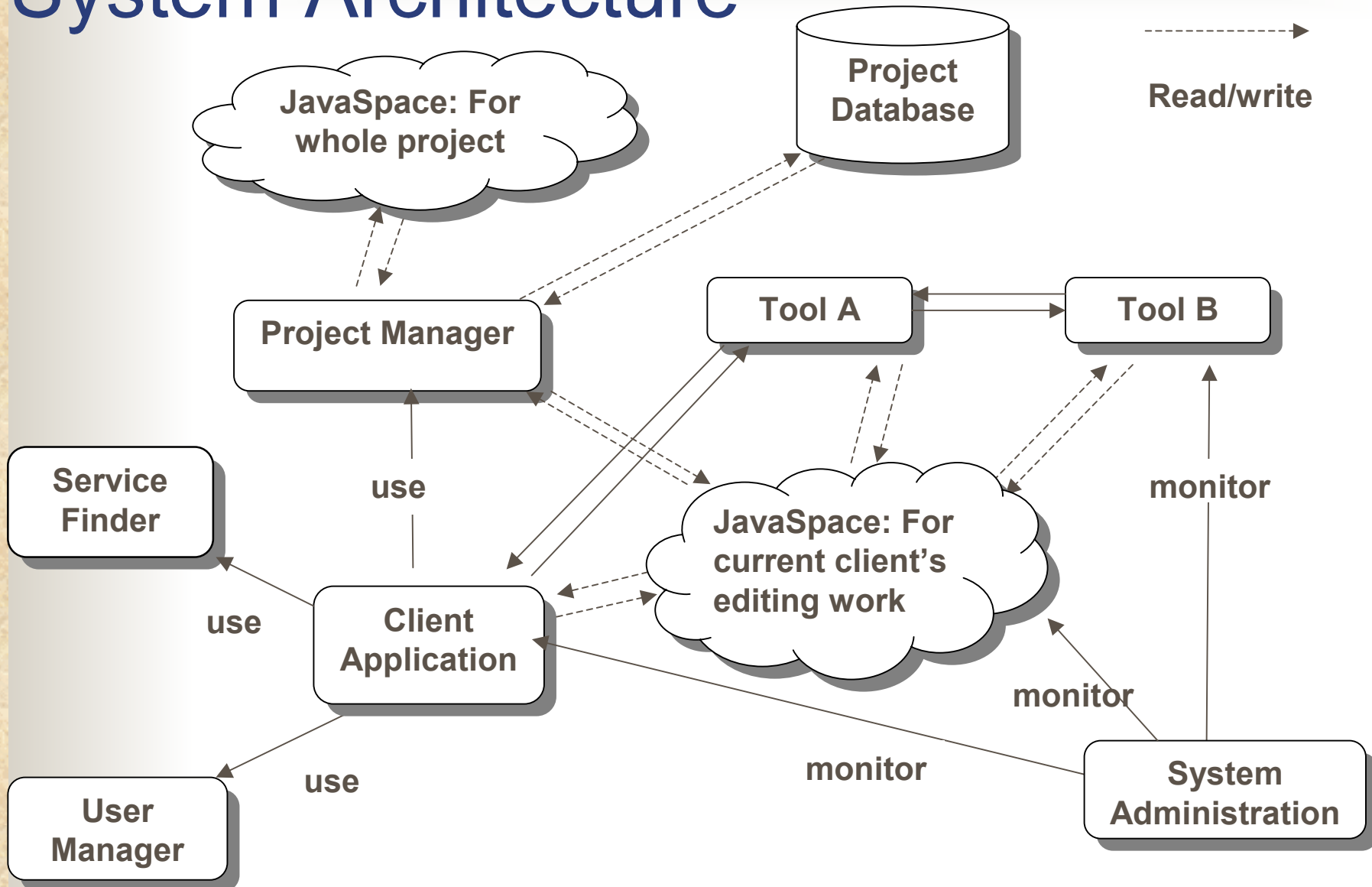
- What?

- Develop an open, distributed software development platform on top of Jini networking technology and making use of JavaSpace technology and design patterns

- How?

- Develop a set of APIs for tool developers to make use of
- Define the communication data structure and protocol for collaboration.

System Architecture





Brief Jini Concept

- Jini is a network technology that enables spontaneous assembly and interaction of services and devices on a network. [Adapted from Jini Network Technology datasheet]
- Provide reliable services in an unreliable environment
 - This includes self-healing by leasing and transaction support for partial failure
- Code mobility which is implemented by notion of Jini proxy
- A Jini system consists of three main parts: Infrastructure, Programming Model and Services



Brief Jini Concept (con't)

- Infrastructure

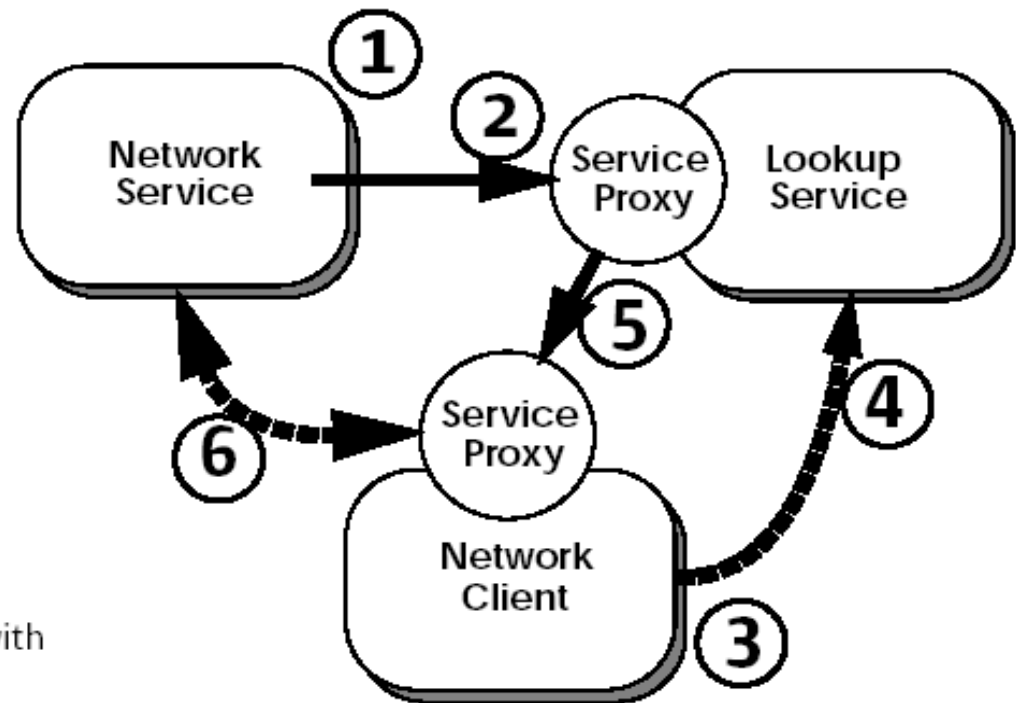
- Discovery, Join and Lookup Protocols
- Lookup Service

- Programming Model

- Leasing
- Remote Event
- Distributed Transaction

Brief Jini Concept (con't)

- 1 Discover**
Network service discovers available lookup services (LUS)
- 2 Join**
Network service sends service proxy to LUS
- 3 Discover**
Network client discovers available LUS
- 4 Lookup**
Network client sends a request to LUS to find desired services
- 5 Receive**
LUS sends registered service proxy to network client
- 6 Use**
Network client interacts directly with network service via service proxy



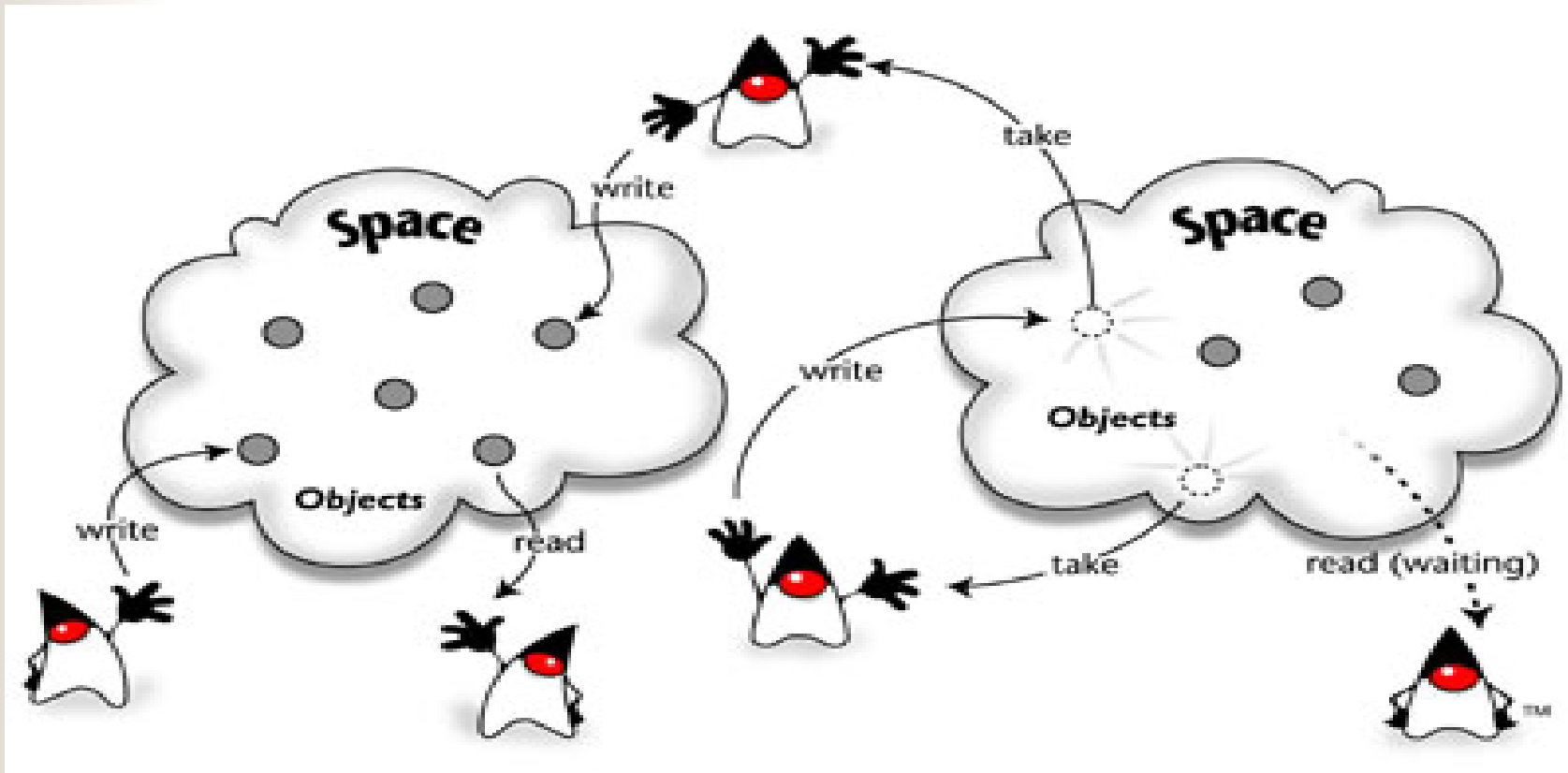
How jini technology works - a flow diagram



Brief JavaSpaces Concept

- Space-based model for distributed application development
- Simple programming model:
 - `read`, `take`, `write` and `notify`

Brief JavaSpaces Concept (con't)

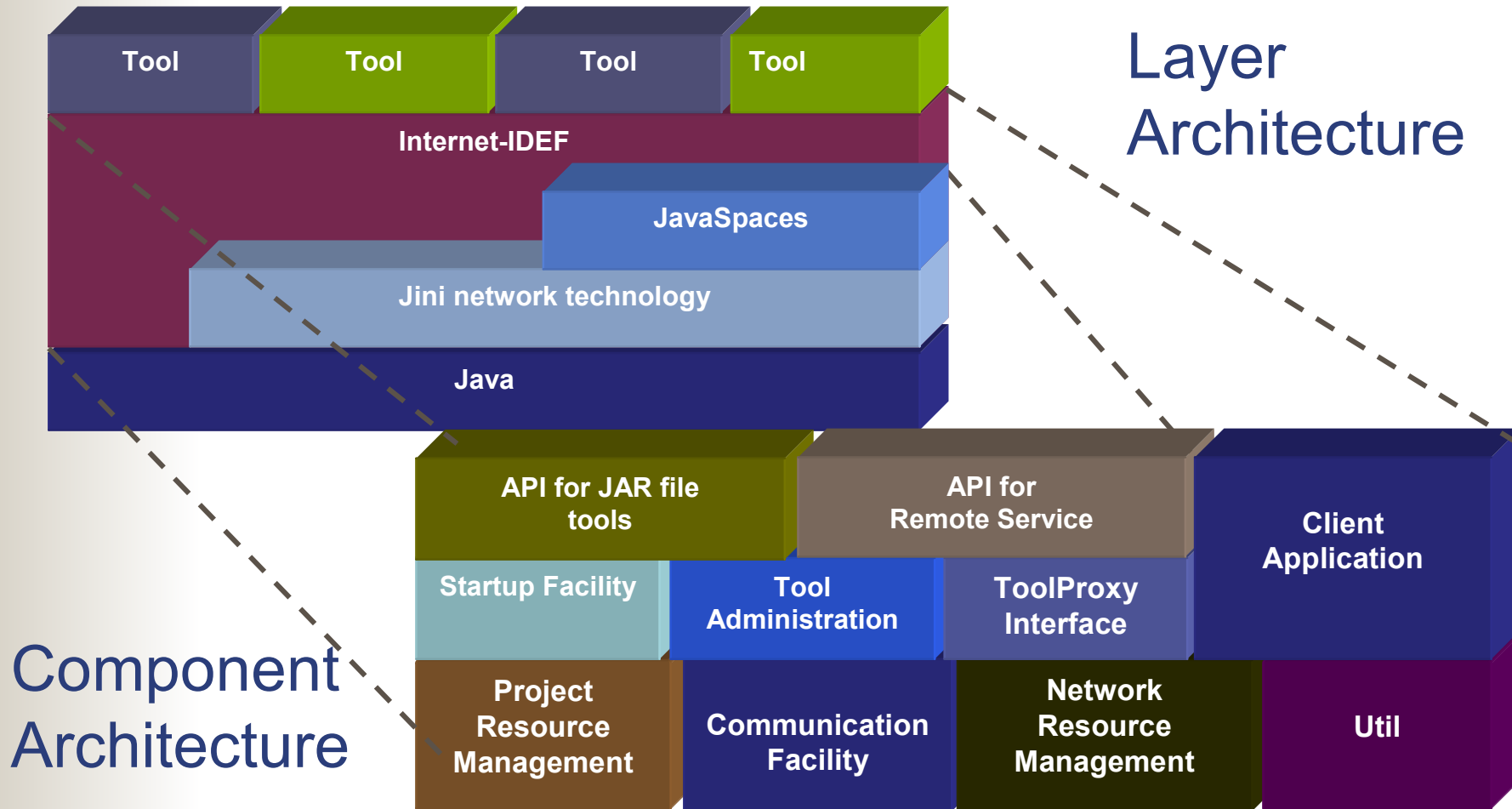




Design and Implementation

- Layered Architecture and Component Architecture
- Jini and Internet-IDEF
- Module API design
- Clients and Tools Collaboration

Layered Architecture and Component Architecture





Jini and Internet-IDEF

■ Problem:

- Jini service lookup based on Java type matching – client application of tools should have knowledge of Tool's "Type" beforehand.
- E.g. Clients only know **Editor** type, they don't know **Compiler** type which is later added to the system

■ Solution:

- A standard **ToolProxy** interface was defined for client applications.



Jini and Internet-IDEF (con't)

- Problem:

- The framework has to support both command line tools and rich GUI tools

- Solution:

- Defined and implemented classes to support both of them (Illustrate later)



Jini and Internet-IDEF (con't)

■ Problem:

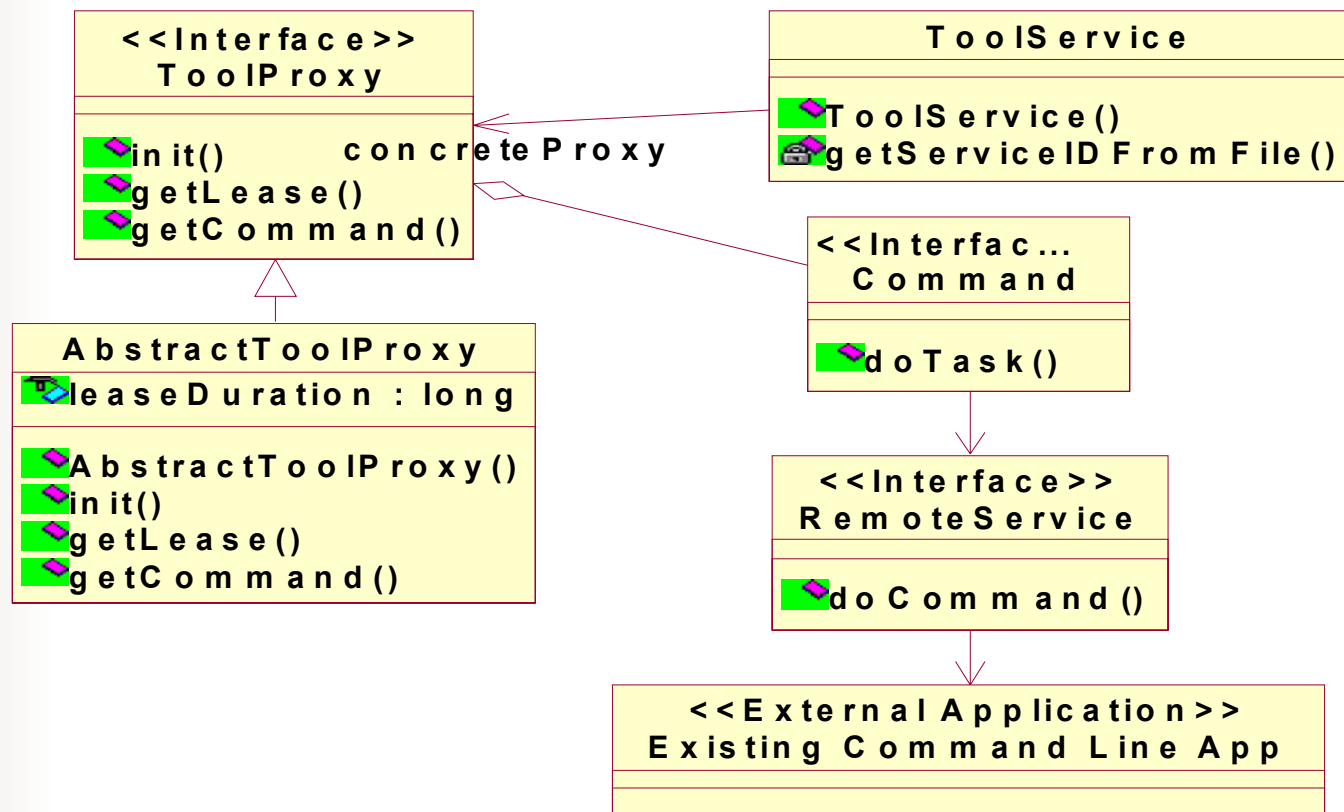
- Repeat Implementation of tool startup steps.
- E.g. Find lookup service, service registration, etc

■ Solution:

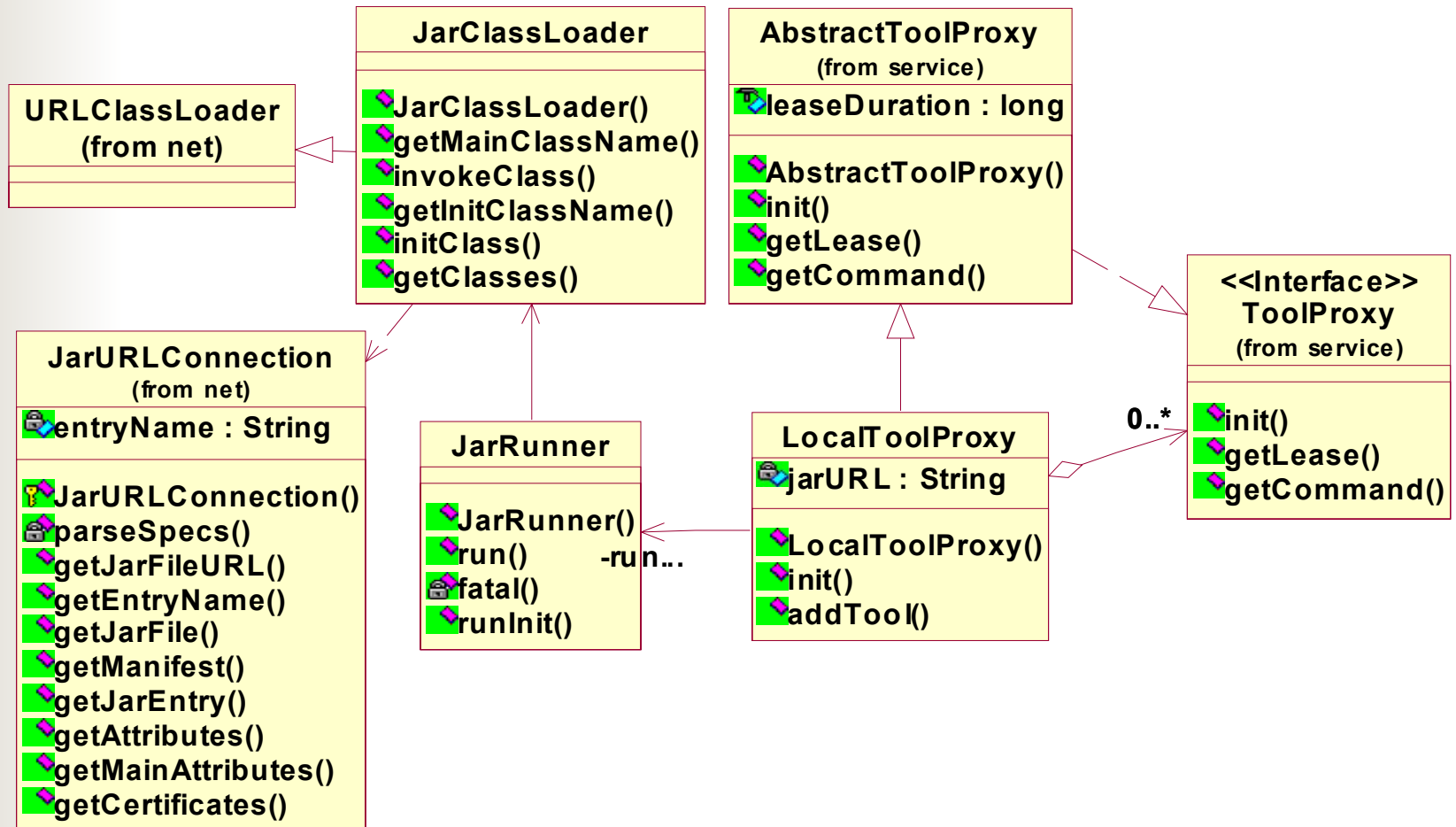
- Implemented a set of classes for standardizing steps of starting up a tool.

Module API design

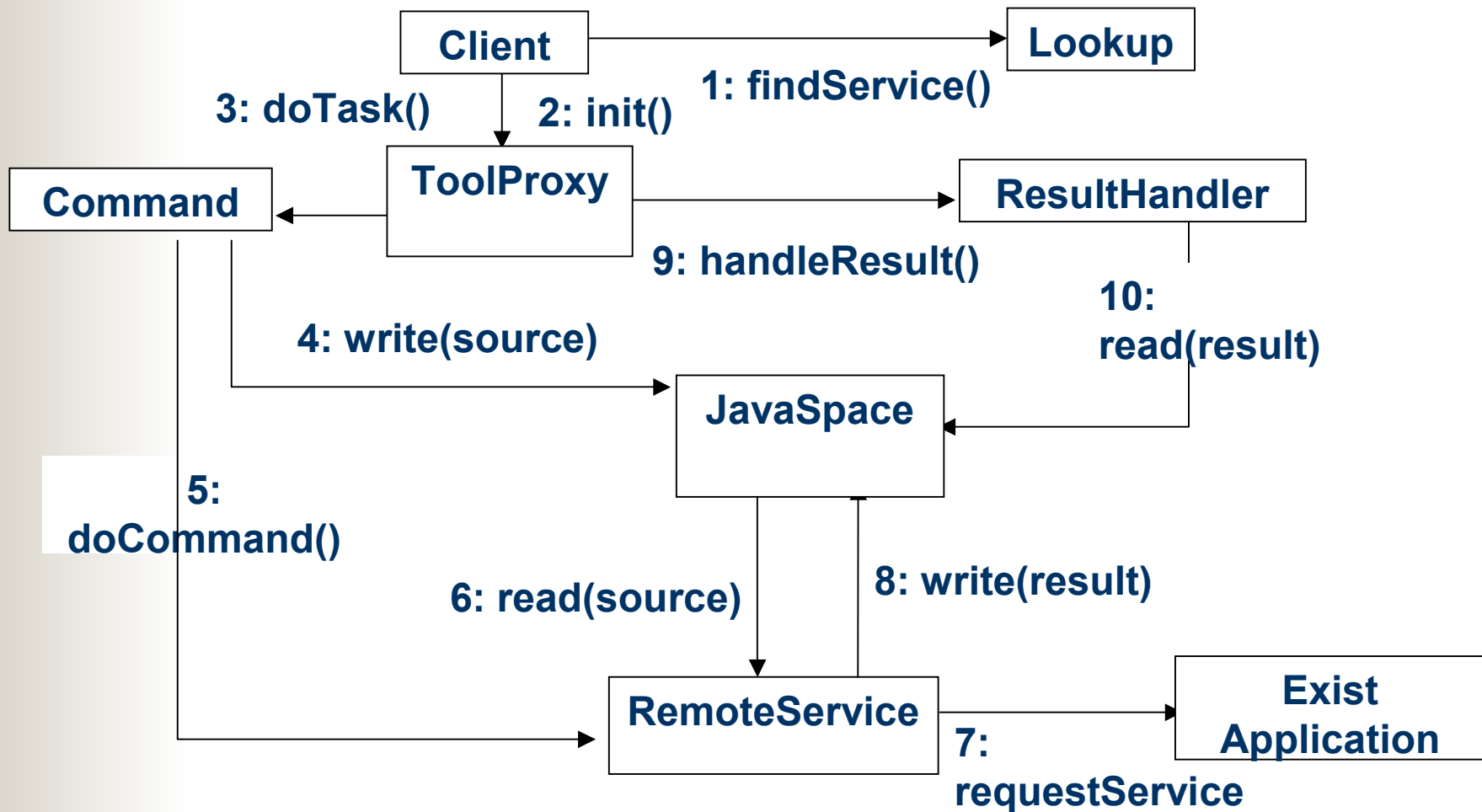
- API for dynamic plug-in of command line tools and JAR applications



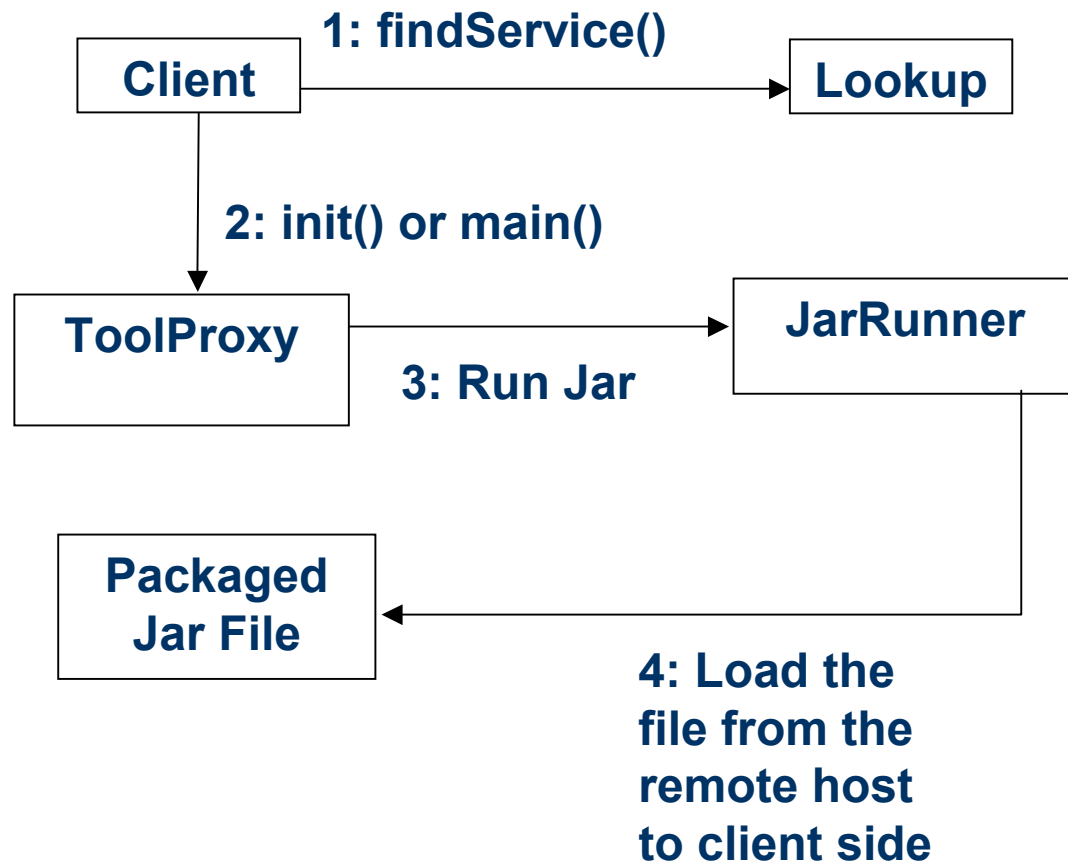
Module API design (con't)



Clients & Tools Collaboration— Command line tool

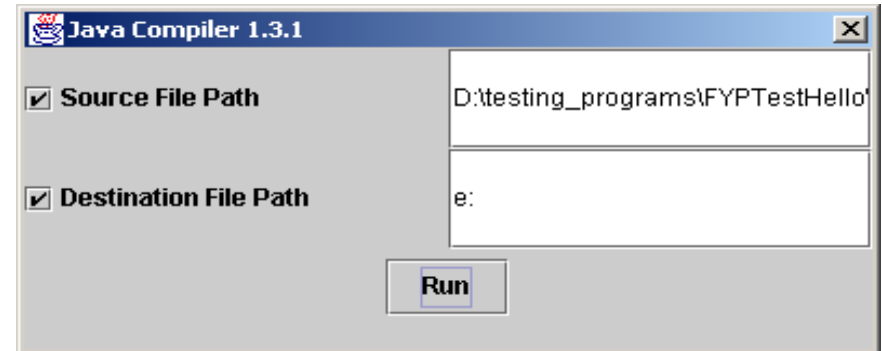
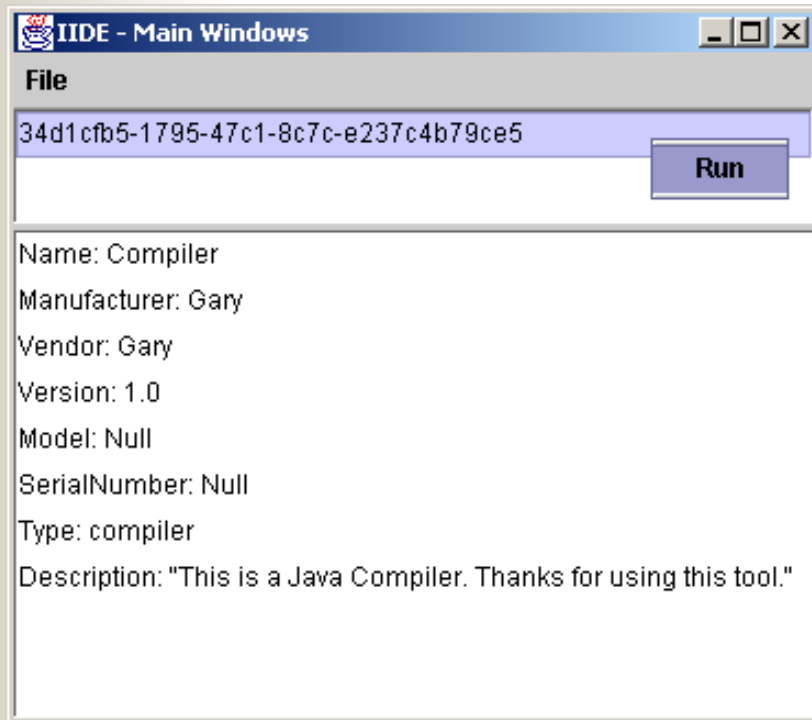


Collaboration Diagram – Rich GUI

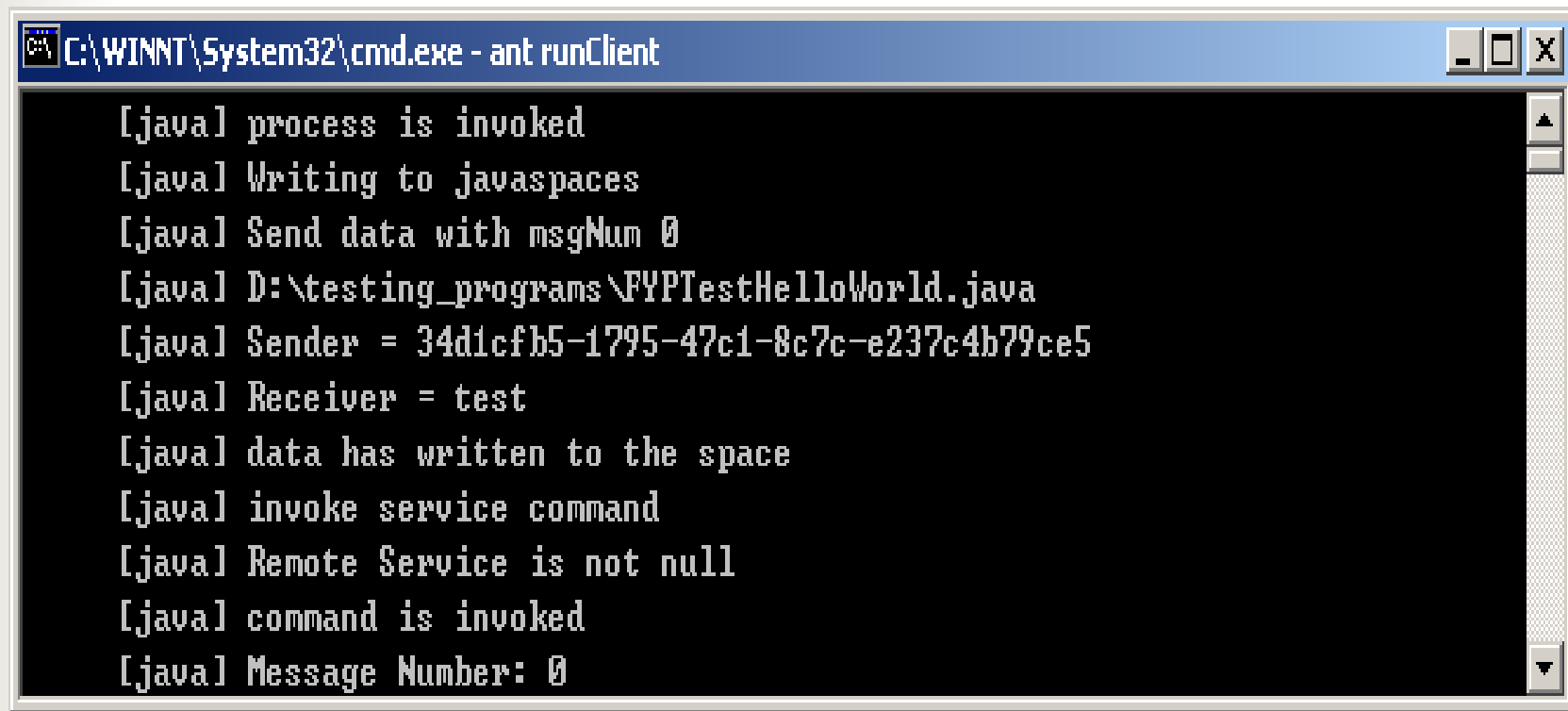


Validation – Remote Java Compiler

- Standalone Java Compiler to Remote Java Compiler



Validation – Remote Java Compiler (con't)



```
C:\WINNT\System32\cmd.exe - ant runClient

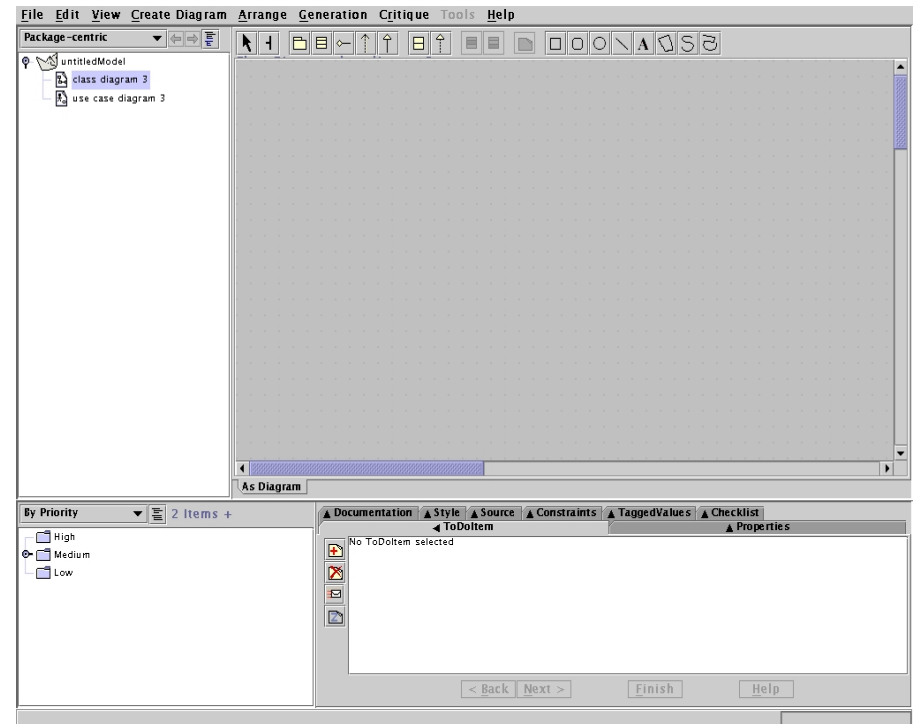
[java] process is invoked
[java] Writing to javaspace
[java] Send data with msgNum 0
[java] D:\testing_programs\FYPTTestHelloWorld.java
[java] Sender = 34d1cfb5-1795-47c1-8c7c-e237c4b79ce5
[java] Receiver = test
[java] data has written to the space
[java] invoke service command
[java] Remote Service is not null
[java] command is invoked
[java] Message Number: 0
```

Validation – Remote Java Compiler (con't)

```
C:\WINNT\System32\cmd.exe - ant runServer
[java] Get Data, Message Num: 0
[java] Template 1 Classes: class fyp.channel.DataObject
[java] class fyp.channel.DataObject
[java] Get Data Test
[java] Template 2 Classes: class fyp.channel.DataObject
[java] Object Taken's Class: class fyp.channel.DataObject
[java] Source: fyp.channel.DataObject@1d75ee
[java] javac c:\temp\FYPTTestHelloWorld.java
[java] c:\temp\FYPTTestHelloWorld.java
[java] Class file pathc:\temp\FYPTTestHelloWorld.class
[java] Compiled
[java] c:\temp\FYPTTestHelloWorld.java
[java] Class file pathc:\temp\FYPTTestHelloWorld.class
[java] File object is written to JavaSpace.
```

Validation – Collaborative UML Editor

- ArgoUML with collaborative capacity
- Version Engine



Validation – Collaborative UML Editor (con't)

The image displays two side-by-side screenshots of the ArgoUML software interface, illustrating the validation process in a collaborative UML editor. Both windows are titled "ArgoUML - Untitled" and show a "Package-centric" view of a model.

Left Screenshot: The diagram shows a class `Car` with attributes `+ speed: int` and `+ increaseSpeed(): void`. It is associated with an interface `<<Interface>> Engine`. Below `Car`, there are two classes: `PrivateCar` and `CarEngin`. `PrivateCar` inherits from `Car` (indicated by a dashed arrow with an open arrowhead), and `CarEngin` inherits from `Engine` (indicated by a dashed arrow with an open arrowhead). A solid line connects `PrivateCar` and `CarEngin`, representing an association.

Right Screenshot: This screenshot shows the same diagram but with several elements highlighted in blue, indicating they are selected. The `Car` class, the `PrivateCar` class, and the `CarEngin` class are all selected. The `Car` class is also highlighted with a red border, suggesting a validation error or warning. The `CarEngin` class has a red wavy line under its name, indicating a missing implementation of the `Engine` interface.

Both screenshots show a "Package-centric" view with a tree on the left containing "untitledModel", "class diagram", "use case diagram", "Car", "CarEngin", "Engine", and "PrivateCar". The bottom of each window features a "By Priority" filter (High, Medium, Low) and a "No ToDoItem selected" message. The interface includes standard menu items (File, Edit, View, Create Diagram, Arrange, Generation, Critique, Tools, Help) and a toolbar.



Challenges

- Making balance between generality and specificity
- Measuring the adaptability of the framework
- Defining an appropriate communication data structure and protocols
- Choosing suitable design patterns



Conclusion

- Developed a set of classes for tool developers to plug in tools to the platform
 - Components implemented
 - Dynamic Plug-in API for remote service and JAR file tools
 - ToolProxy interface
 - Service Utilities
 - Communication Facility
 - Tool Administration
 - Client tools browser
 - Tool Startup Facility



Conclusion (con't)

- Plug-in a Java compiler and collaborative UML editor
 - Operate successfully
- Successful integration of these tools into the framework validates the proposed design



Potential enhancements

- Combination with other technologies, e.g.
 - Java Bean
 - XML
- Remain components, e.g.
 - Project resource management component
 - User management component



Thank You

Q & A Session