Frameworks

- Aims of this section:
  - Look at the notion of frameworks
  - Explore two frameworks supporting software tool development – Eclipse and Argo (see the ArgoMTE handout paper)
- Later
  - Look at Pattern Languages
    - collections of patterns that used together lead to solutions for a particular domain area
  - Illustrate with a pattern language for developing frameworks together with its use in the evolution of MViews/JViews for software tool construction

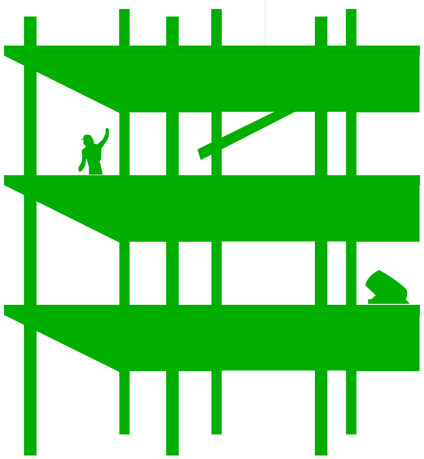
Frameworks

- "A framework is a set of classes that embodies an abstract design for solutions to a family of related problems"
  - Ralph Johnson, "Designing Reusable Classes", The Journal of Object-Oriented Programming, Vol.1,No.2, 1988, pp 22-35
- "A software framework is a reusable mini-architecture that provides the generic structure and behavior for a family of software abstractions, along with a context of memes/metaphors which specifies their collaboration and use within a given domain."
  - Brad Appleton "Patterns and Software: Essential Concepts and Terminology"
- Provide a prefrabricated structure or template for applications in a particular domain
  - eg an application framework provides the support for "default" behaviour for drawing windows, scollbars and menus
  - "Leveraging Object-Oriented Frameworks" Taligent white paper

    http://www.ibm.com/java/education/ooleveraging/index.html

Examples of frameworks

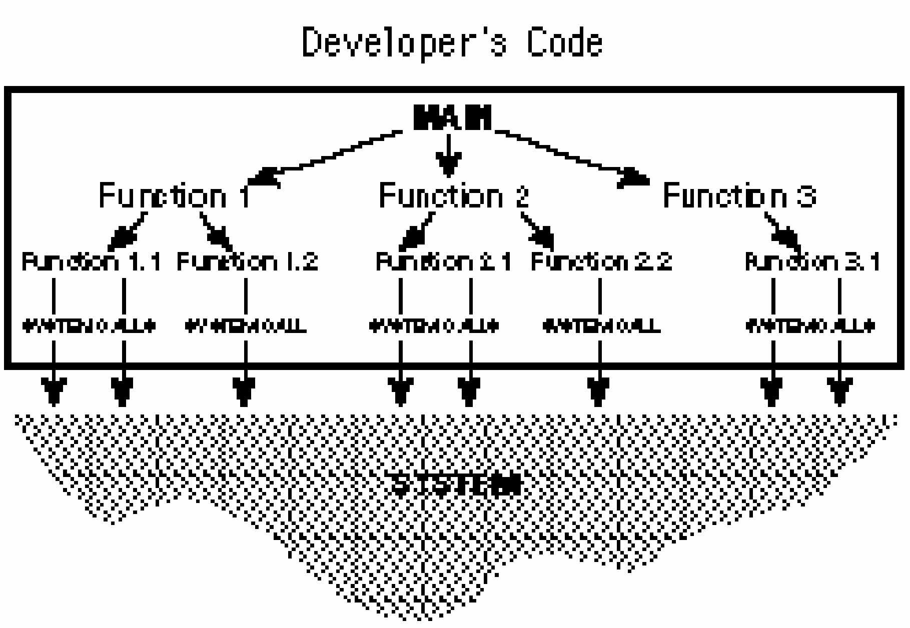
- Many of the Java APIs are frameworks for developing applications or applets for a particular domain
  - eg AWT, Swing for GUI applications
- Many IDEs provide application development frameworks
  - eg Eclipse, Argo UML, Visual Studio, ArchStudio
- Some widely successful and influential frameworks include:
  - ObjectTime
  - Unidraw/HotDraw
  - ET++
  - MVC
  - MacApp
  - IBM's Spring (for Java)



Framework vs procedural and OOP

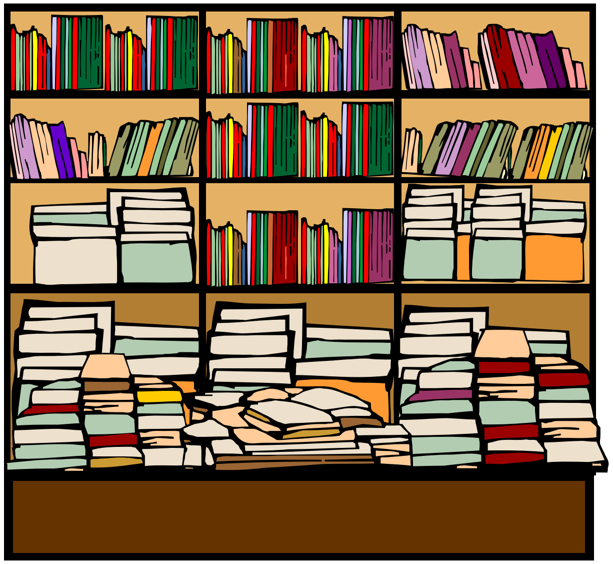
- Procedural
  - Developers code calls the "system" code via library calls
  - Developer responsible for overall behaviour and flow of control
  - system code provides underlying functionality
- Problems
  - difficult to extend "system"
  - difficult to factor common code

Figure 1



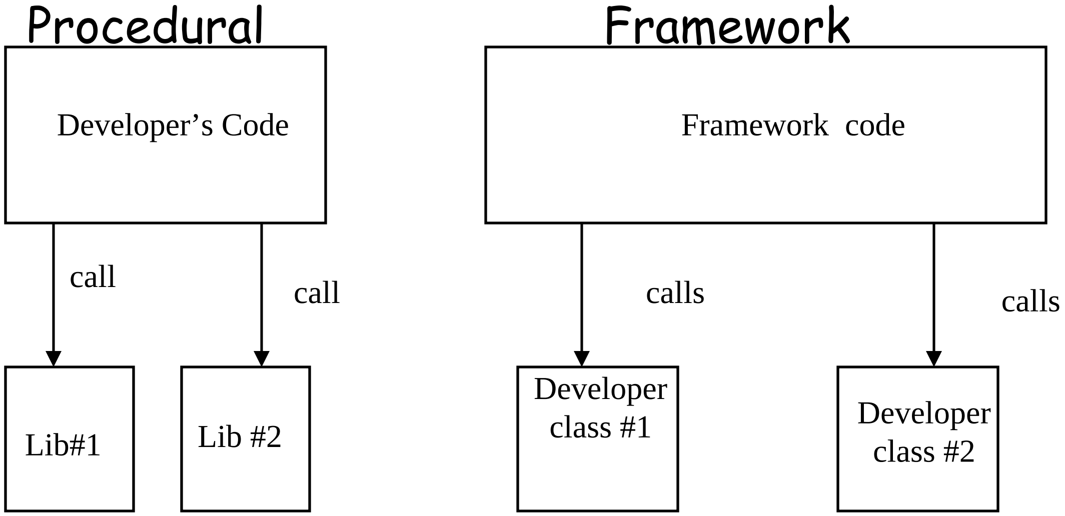
OOP and class libraries

- An improvement in terms of factoring out common code and improving maintainability
- But developer still responsible for the main program flow
  - client instantiates classes from class library
  - client calls functions
  - little predefined flow of control or interaction
  - little default behaviour



Framework oriented programming

- Frameworks provide infrastructure and design
  - basic flow of control and internal structure “wired” in
- The framework calls the developers code (Hollywood principle – “don’t call us, we’ll call you…”)
  - roles reversed compared with procedural programming
  - Eg Applets in Java

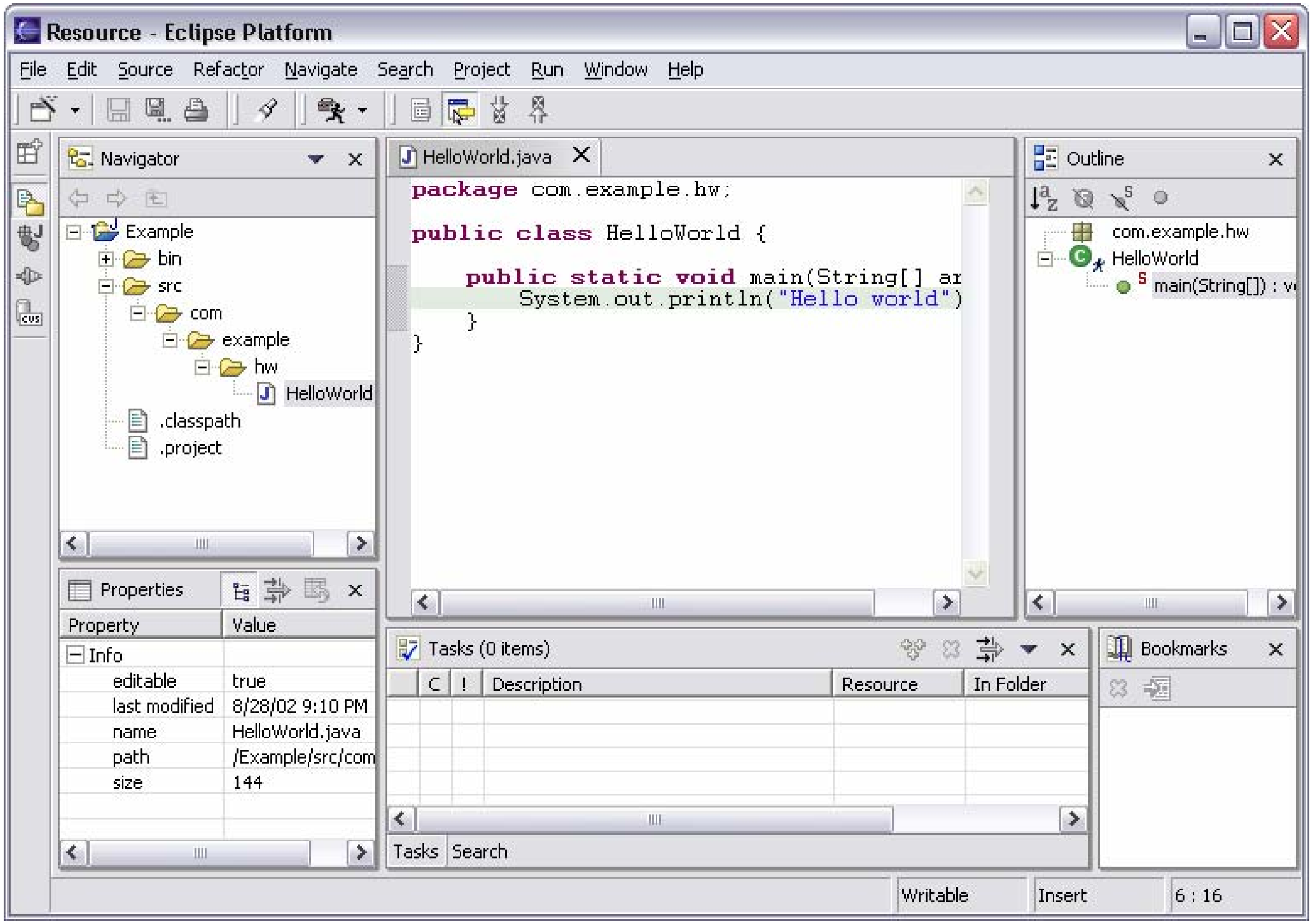


Eclipse

- Project Aims:
    - Provide open platform for application development tools
        - Run on a wide range of operating systems
        - GUI and non-GUI
    - Language-neutral
        - Permit unrestricted content types
        - HTML, Java, C, JSP, EJB, XML, GIF, …
    - Facilitate seamless tool integration
        - At UI and deeper
        - Add new tools to existing installed products
    - Attract community of tool developers
        - Including independent software vendors (ISVs)
        - Capitalize on popularity of Java for writing tools

- Material in this section from http://eclipse.org/eclipse/
    - (abridged version of slideset from this site)

Example

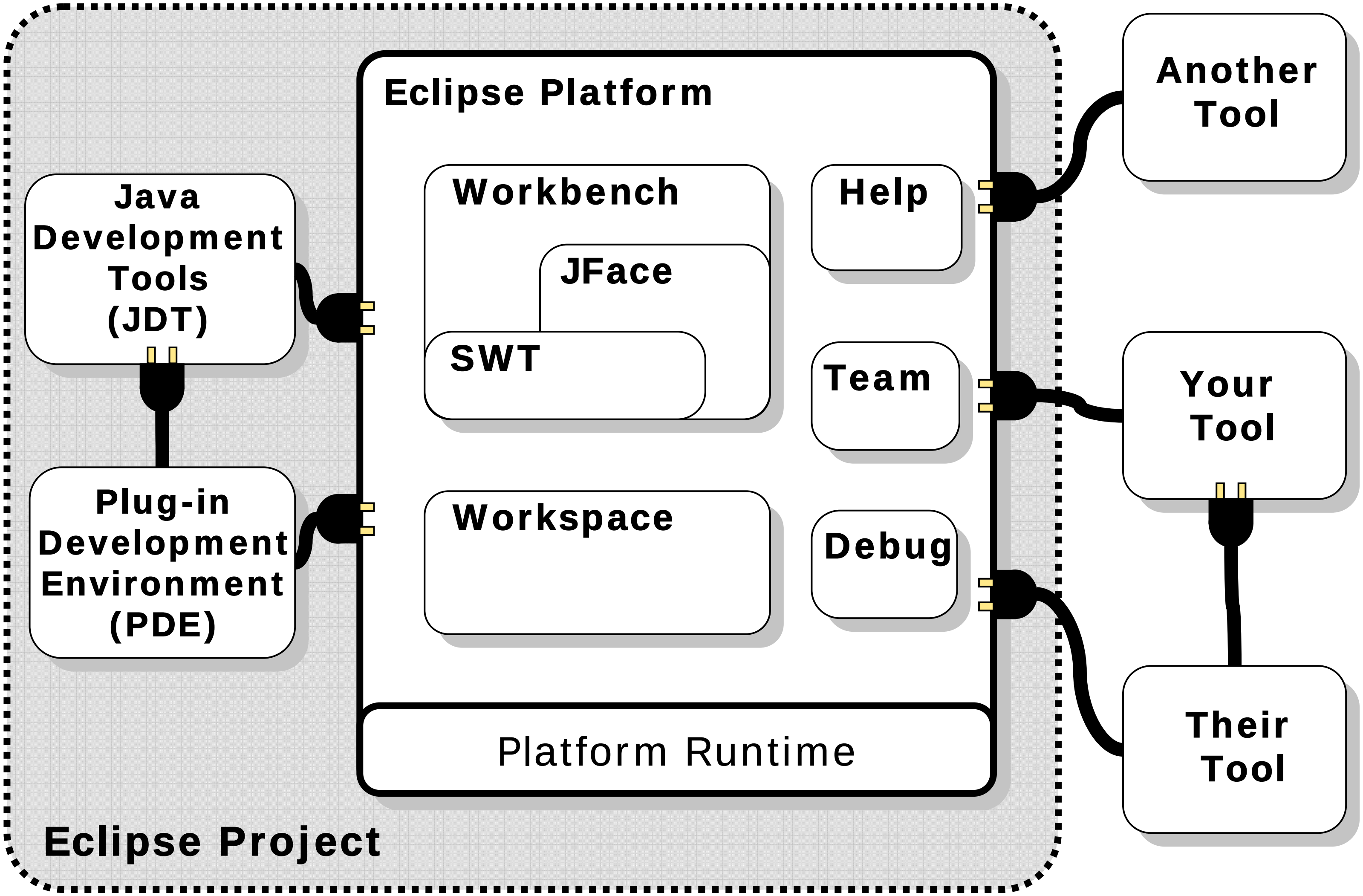


```
package com.example.hw;

public class HelloWorld {

    public static void main(String[] ar
        System.out.println("Hello world")
    }
}
```

Property	Value
Info	
editable	true
last modified	8/28/02 9:10 PM
name	HelloWorld.java
path	/Example/src/com
size	144

C	!	Description	Resource	In Folder

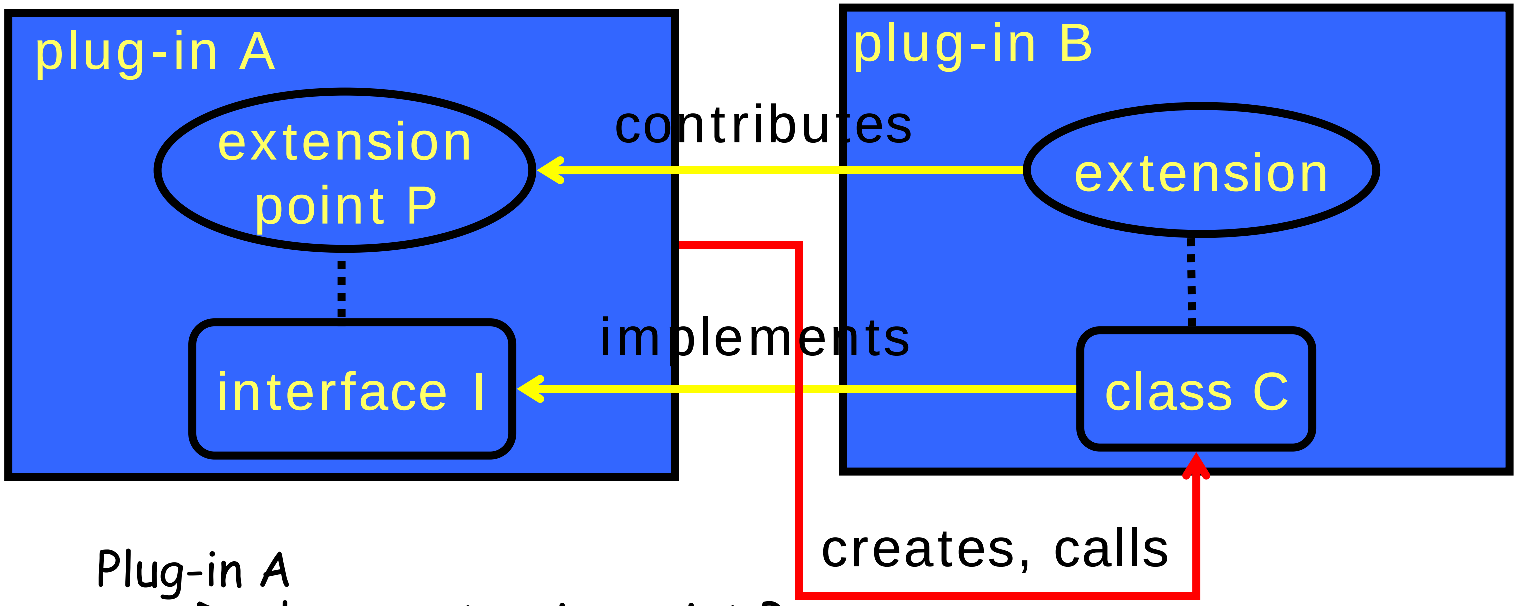
Architectural overview



Plug in approach

- Plug-in - smallest unit of Eclipse function
  - Big example: HTML editor
  - Small example: Action to create zip files
- Extension point - named entity for collecting "contributions"
  - Example: extension point for workbench preference UI
- Extension - a contribution
  - Example: specific HTML editor preferences
- Each plug-in
  - Contributes to 1 or more extension points
  - Optionally declares new extension points
  - Depends on a set of other plug-ins
  - Contains Java code libraries and other files
  - May export Java-based APIs for downstream plug-ins
  - Lives in its own plug-in subdirectory
- Details spelled out in the plug-in manifest (XML)

Example



Plug-in A

Declares extension point P

Declares interface I to go with P

Plug-in B

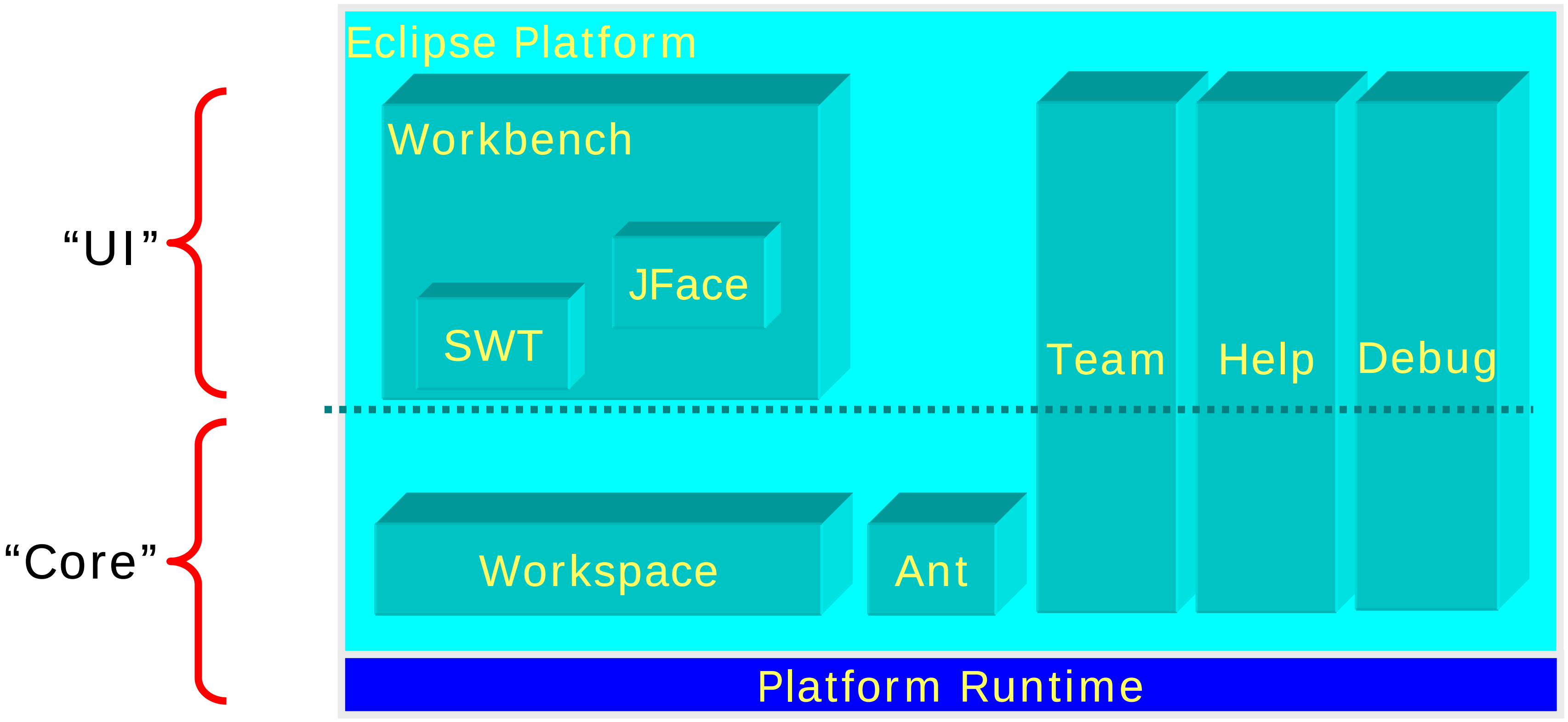
Implements interface I with its own class C

Contributes class C to extension point P

Plug-in A instantiates C and calls its I methods

Eclipse Platform

- Eclipse Platform is the common base
- Consists of several key components



“UI”

“Core”

Eclipse Platform

Workbench

SWT

JFace

Team

Help

Debug

Workspace

Ant

Platform Runtime

Workspace

- Manages projects which user is working on
- Projects consist of resources (eg source files, folders, projects) in a tree construct
  - Tools read, create, modify, and delete resources in workspace
- Plug-ins access via workspace and resource APIs
  - Allows fast navigation of workspace resource tree
  - Resource change listener for monitoring activity
  - Resource deltas describe batches of changes
  - Maintains limited history of changed/deleted files
  - Several kinds of extensible resource metadata
  - Workspace session lifecycle
  - Incremental project builders
    - Plugins to manage analysis & compilation (eg Java Builder in JDT)

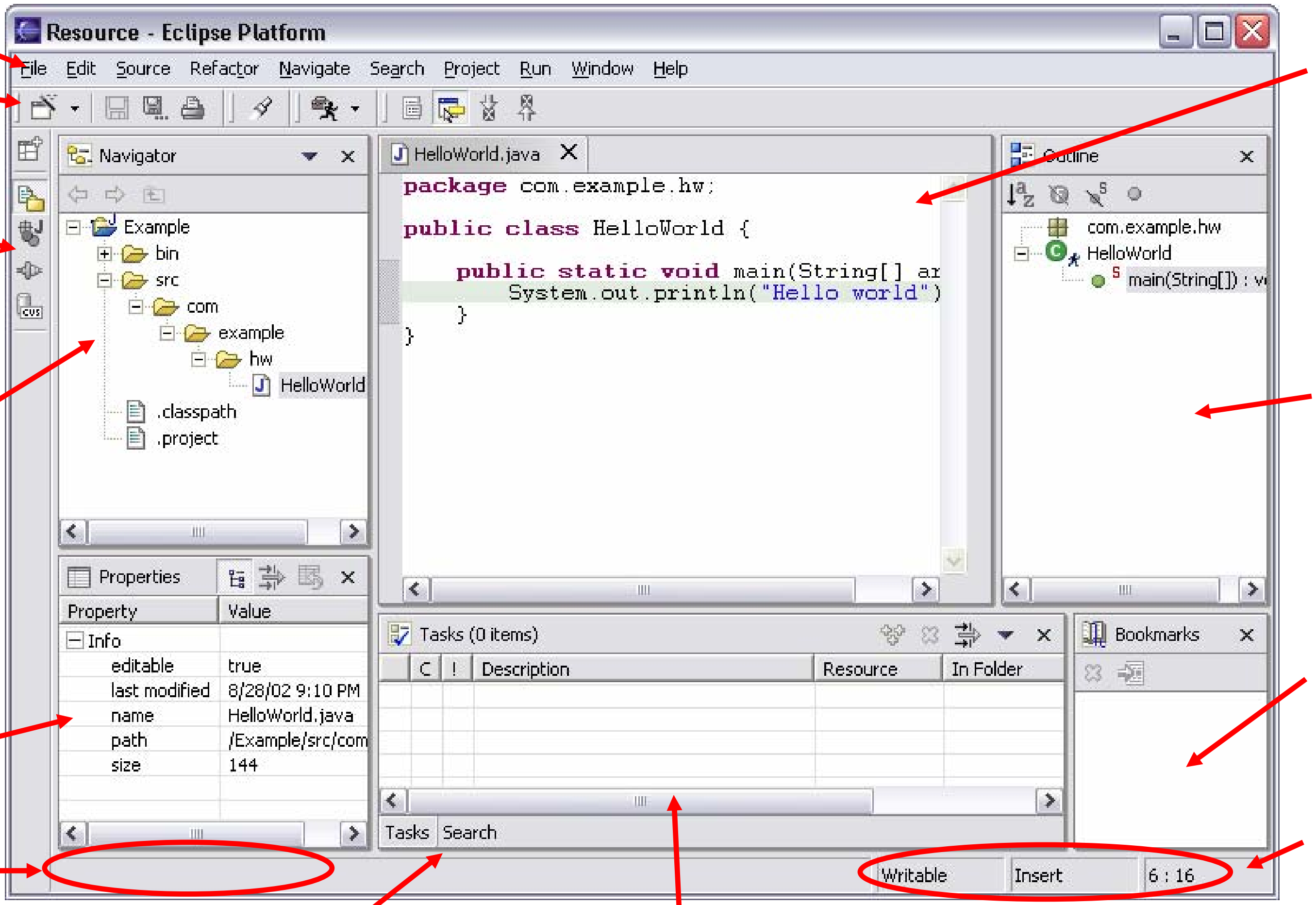
Workbench

- SWT – generic low-level graphics and widget set
  - Generic graphics and GUI widget set
  - OS-independent API
  - Uses native widgets where available, emulates otherwise
- JFace – UI frameworks for common UI tasks
  - Classes for handling common UI tasks
  - API and implementation are window-system independent
- Workbench – UI personality of Eclipse Platform, centred on:
  - Editors
  - Views
  - Perspectives

Workbench

- Editors appear in workbench editor area
  - Contribute actions to workbench menu and tool bars
  - Open, edit, save, close lifecycle
  - Extension point for contributing new types of editors
    - Eg: JDT provides Java source file editor
  - Eclipse Platform includes simple text file editor
- Views provide information on some object
  - By augmenting:
    - Editors, eg: Outline view summarizes content
    - Other views, eg: Properties view describes selection
  - Eclipse Platform includes many standard views: Resource Navigator, Outline, Properties, Tasks, Bookmarks, Search, …
- Perspectives are arrangements of views and editors
  - Different perspectives suited for different user tasks
  - Users can quickly switch between perspectives
  - Eclipse Platform includes standard perspectives: Resource, Debug, …

Workbench in use



```
package com.example.hw;

public class HelloWorld {

    public static void main(String[] ar
        System.out.println("Hello world")
    }
}
```

Property	Value
Info	
editable	true
last modified	8/28/02 9:10 PM
name	HelloWorld.java
path	/Example/src/com
size	144

C	!	Description	Resource	In Folder

Menu bar

Tool bar

Perspective and Fast View bar

Resource Navigator view

Properties view

Message area

Stacked views

Tasks view

Text editor

Outline view

Bookmarks view

Editor Status area

Other components

- Team
  - Version and configuration management (VCM)
  - Share resources with team via a repository (project level assocn)
  - Eclipse Platform includes CVS repository provider
- Debug
  - Common debug UI and underlying debug model
- Help
  - Help books are HTML webs presented in standard web browser
  - Help mechanisms available to all plug-ins
  - Help search engine based on Apache Lucene
- Ant
  - Eclipse incorporates Apache Ant
  - Run Ant targets in build files inside or outside workspace
  - PDE uses Ant for building deployed form of plug-in

Platform Summary

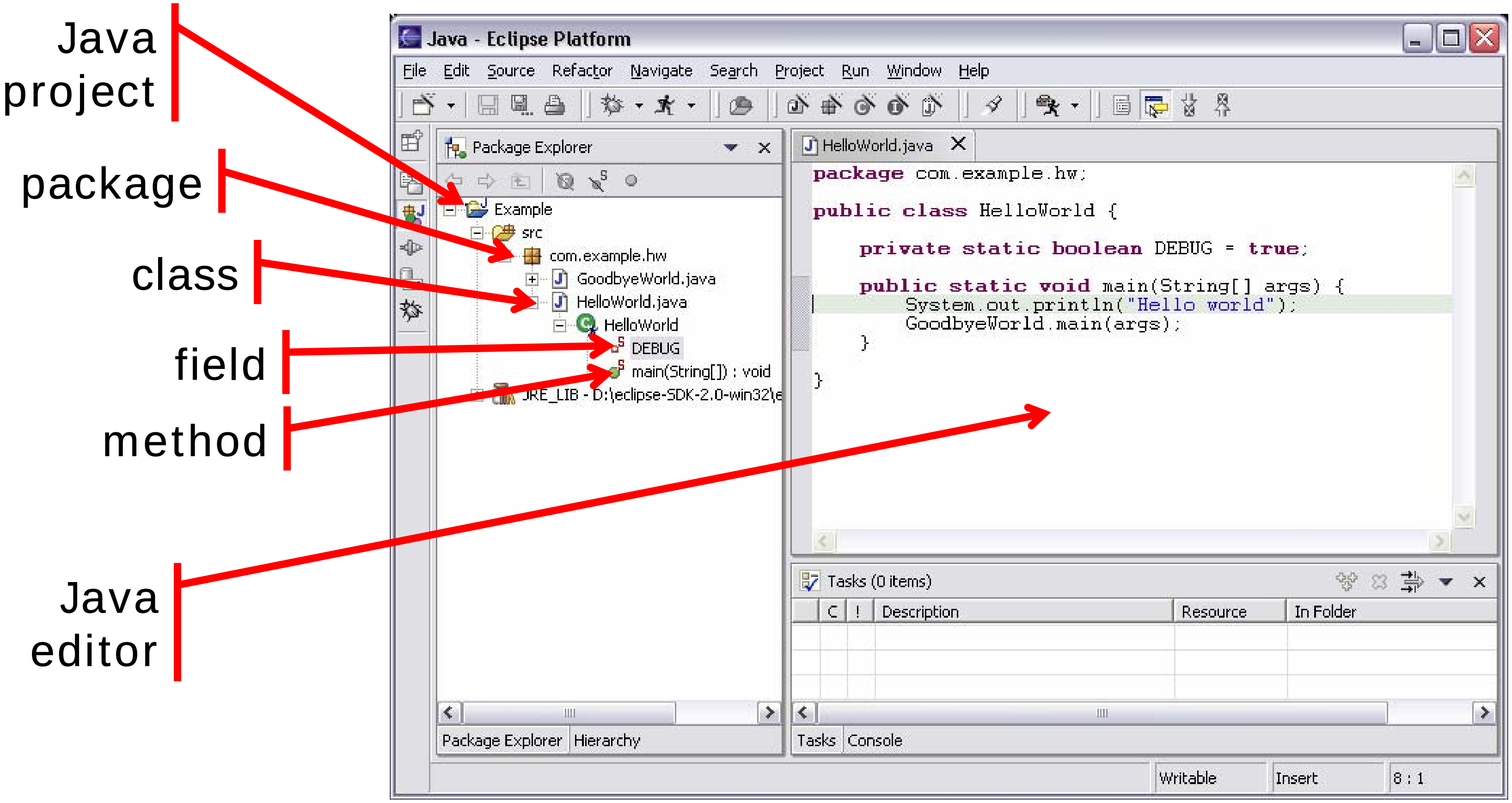
- Eclipse Platform provides the nucleus for IDE products
- Plug-ins, extension points, extensions
  - Open, extensible architecture
- Workspace, projects, files, folders
  - Common place to organize & store development artifacts
- Workbench, editors, views, perspectives
  - Common user presentation and UI paradigm
- Key building blocks and facilities
  - Help, team support, internationalization, …

JDT - Example Eclipse toolset

- Java development environment
- Built on top of Eclipse Platform
  - Implemented as Eclipse plug-ins
  - Using Eclipse Platform APIs and extension points
- Included in Eclipse Project releases

Provides Java Perspective

- Java-centric view of files in Java projects



Java project

package

class

field

method

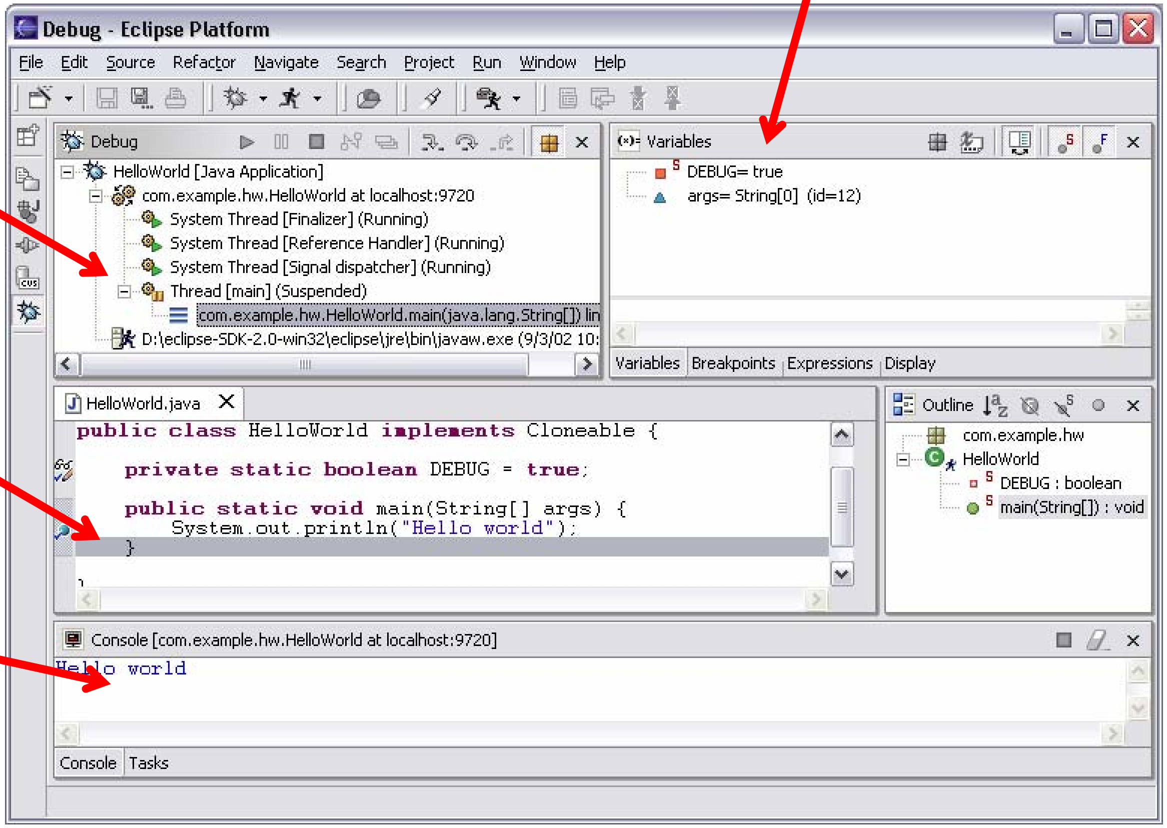
Java editor

```
package com.example.hw;

public class HelloWorld {

    private static boolean DEBUG = true;

    public static void main(String[] args) {
        System.out.println("Hello world");
        GoodbyeWorld.main(args);
    }

}
```

Other features

- Move up & down type hierarchies (super <-> sub class)
- Search for elements
- Javadoc tool tips
- Method signature completion suggestions
- Java specific spellcheck and correction suggestion
- Code templates and stub method creation
- Critiquing tools (eg identifier name suggestions)
- Code refactoring
- Java Compiler

Java debugger


Debug - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Debug
Variables
Variables Breakpoints Expressions Display
HelloWorld.java
Outline
Console Tasks

```
HelloWorld [Java Application]
  com.example.hw.HelloWorld at localhost:9720
    System Thread [Finalizer] (Running)
    System Thread [Reference Handler] (Running)
    System Thread [Signal dispatcher] (Running)
    Thread [main] (Suspended)
      com.example.hw.HelloWorld.main(java.lang.String[]) lin
  D:\eclipse-SDK-2.0-win32\eclipse\jre\bin\javaw.exe (9/3/02 10:
```

```
DEBUG= true
args= String[0] (id=12)
```

```
public class HelloWorld implements Cloneable {

    private static boolean DEBUG = true;

    public static void main(String[] args) {
        System.out.println("Hello world");
    }
}
```

```
com.example.hw
HelloWorld
  DEBUG : boolean
  main(String[]) : void
```

Console [com.example.hw.HelloWorld at localhost:9720]

```
Hello world
```

Local variables

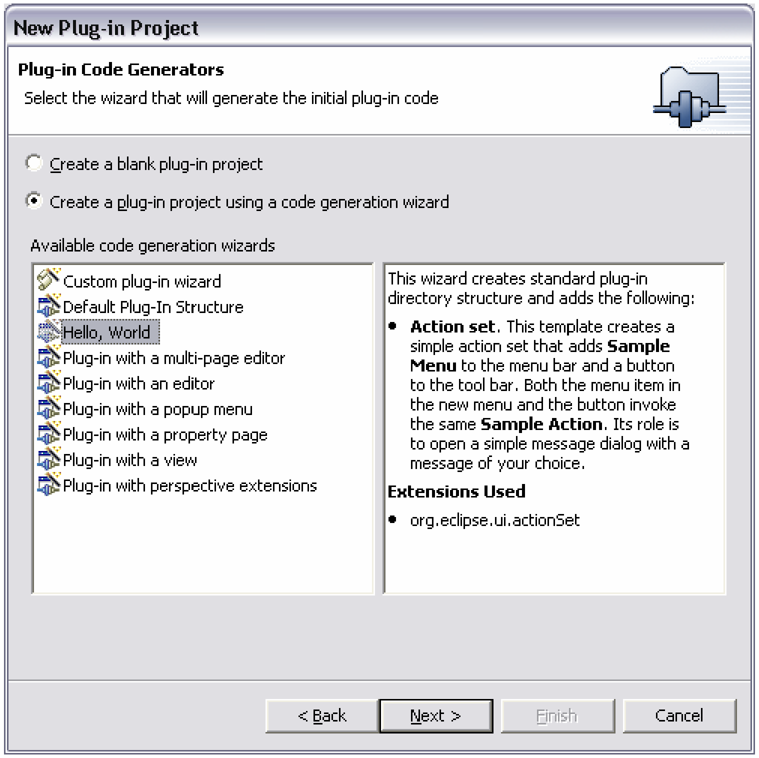
Threads and stack frames

Editor with breakpoint marks

Console I/O

Plugin Development Environment PDE

- Specialized tools for developing Eclipse plug-ins
- PDE templates for creating simple plug-in projects
- Specialized PDE editor for plug-in manifest files



New Plug-in Project

Plug-in Code Generators

Select the wizard that will generate the initial plug-in code

○ Create a blank plug-in project

● Create a plug-in project using a code generation wizard

Available code generation wizards

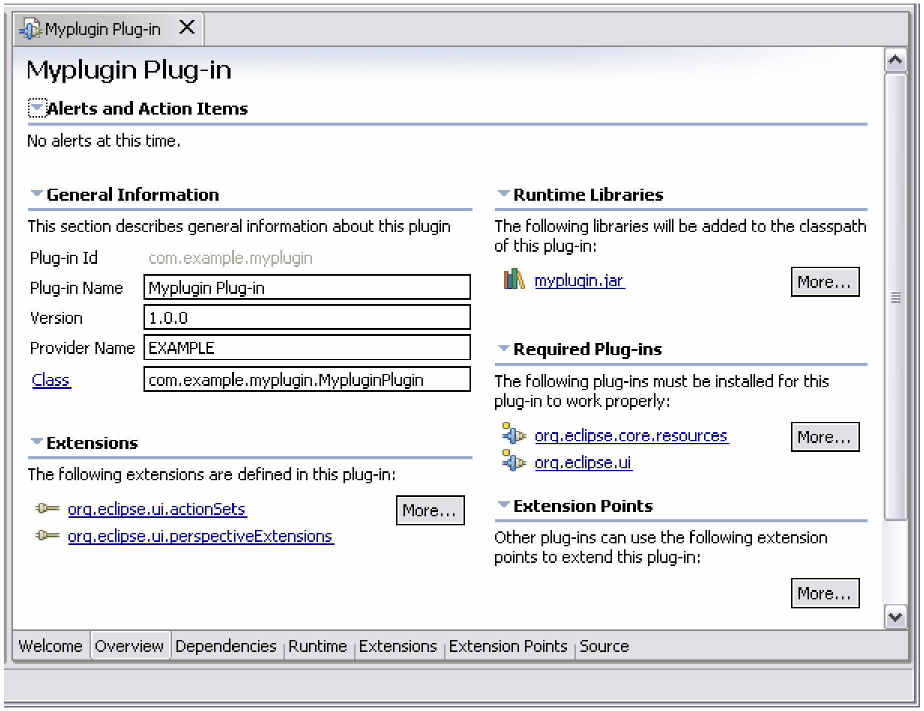
- Custom plug-in wizard
- Default Plug-In Structure
- Hello, World
- Plug-in with a multi-page editor
- Plug-in with an editor
- Plug-in with a popup menu
- Plug-in with a property page
- Plug-in with a view
- Plug-in with perspective extensions

This wizard creates standard plug-in directory structure and adds the following:

- **Action set**. This template creates a simple action set that adds **Sample Menu** to the menu bar and a button to the tool bar. Both the menu item in the new menu and the button invoke the same **Sample Action**. Its role is to open a simple message dialog with a message of your choice.

Extensions Used

- org.eclipse.ui.actionSet



Myplugin Plug-in

Alerts and Action Items

No alerts at this time.

General Information

This section describes general information about this plugin

Plug-in Id: com.example.myplugin

Plug-in Name: Myplugin Plug-in

Version: 1.0.0

Provider Name: EXAMPLE

Class: com.example.myplugin.MypluginPlugin

Extensions

The following extensions are defined in this plug-in:

- org.eclipse.ui.actionSets
- org.eclipse.ui.perspectiveExtensions

Runtime Libraries

The following libraries will be added to the classpath of this plug-in:

- myplugin.jar

Required Plug-ins

The following plug-ins must be installed for this plug-in to work properly:

- org.eclipse.core.resources
- org.eclipse.ui

Extension Points

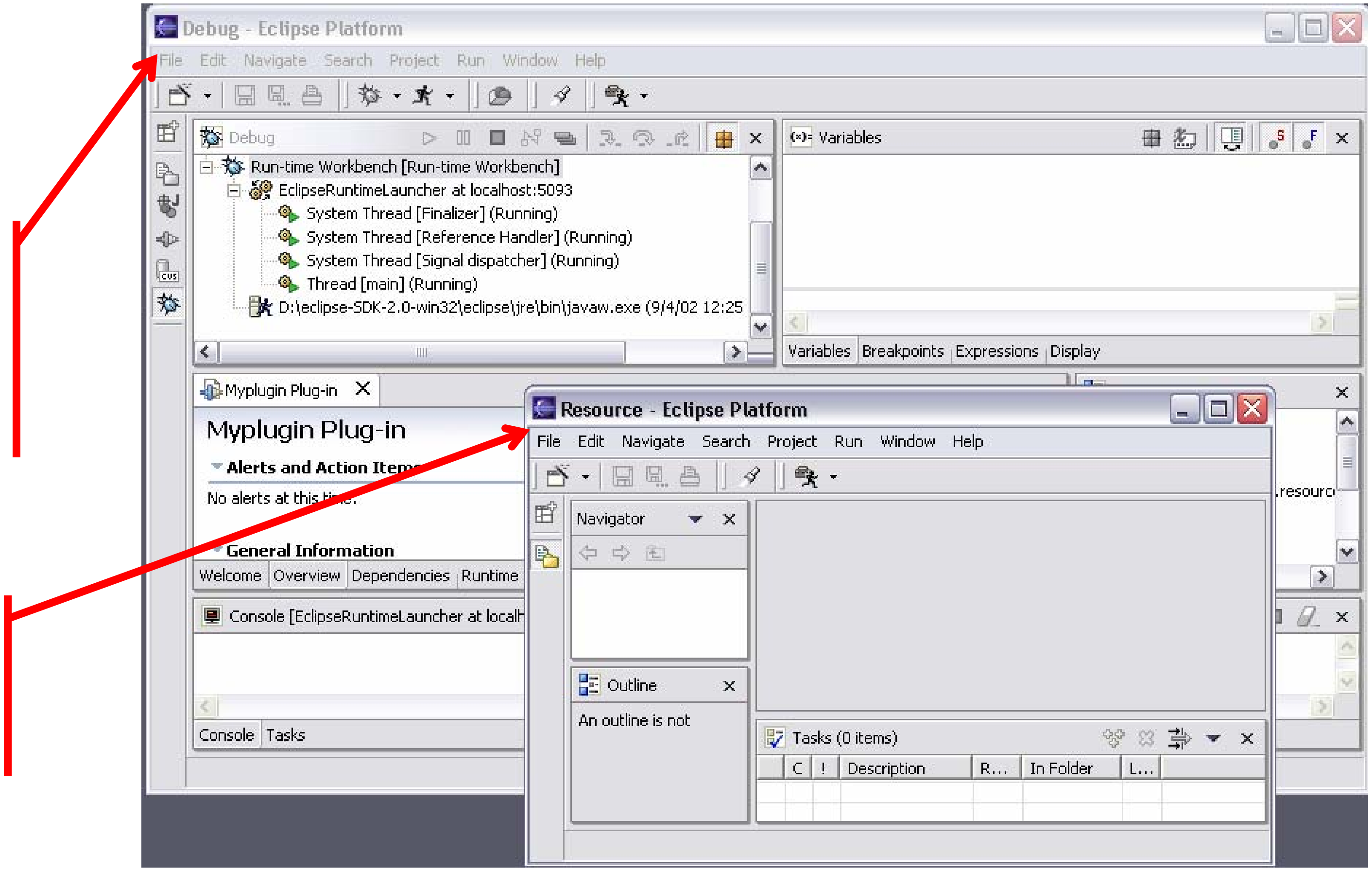
Other plug-ins can use the following extension points to extend this plug-in:

PDE

- **PDE runs and debugs another Eclipse workbench**

1. Workbench running PDE (host)

2. Run-time workbench (target)



Debug - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Debug
Variables

```
Run-time Workbench [Run-time Workbench]
  EclipseRuntimeLauncher at localhost:5093
    System Thread [Finalizer] (Running)
    System Thread [Reference Handler] (Running)
    System Thread [Signal dispatcher] (Running)
    Thread [main] (Running)
  D:\eclipse-SDK-2.0-win32\eclipse\jre\bin\javaw.exe (9/4/02 12:25
```

Variables Breakpoints Expressions Display
Myplugin Plug-in

Myplugin Plug-in

Alerts and Action Items

No alerts at this time.

General Information

Welcome Overview Dependencies Runtime
Console [EclipseRuntimeLauncher at localh
Console Tasks
.resourc
Resource - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Navigator
Outline

An outline is not

C	!	Description	R...	In Folder	L...

Lessons from Eclipse

- Rules for Enablers from Kent Beck's "Contributing to Eclipse"
- Invitation Rule - Whenever possible, let others contribute to your contributions.
- Lazy Loading Rule - Contributions are only loaded when they are needed.
- Safe Platform Rule - As the provider of an extension point, you must protect yourself against misbehavior on the part of extenders.
- Fair Play Rule - All clients play by the same rules, even me.
- Explicit Extension Rule - Declare explicitly where a platform can be extended.
- Diversity Rule - Extension points accept multiple extensions.
- Good Fences Rule - When passing control outside your code, protect yourself.
- Explicit API Rule - separate the API from internals.
- Stability Rule - Once you invite someone to contribute, don't change the rules.
- Defensive API Rule - Reveal only the API in which you are confident, but be prepared to reveal more API as clients ask for it.

Eclipse summary

- Eclipse has very rapidly developed significant momentum
  - See plugin site for list of commercial and open source plugins
    - http://eclipse.org/community/plugins.html
- Reasons for success
  - Plenty of basic support for tool building from framework
    - Enough stuff "for free" to overcome inertia of understanding the model and working within it
  - Plugin approach is highly successful
    - Principled enough to allow many plugins to collaborate
    - But has issues with informality of spec (see Dietrich et al paper)
  - Open source, but allows commercial extension
- Problems
  - A LOT of things to get your head around if you are starting out developing a plugin
    - Need for more high level support tools to assist in Eclipse tool development (see Marama and EFPL lecture)

Argo

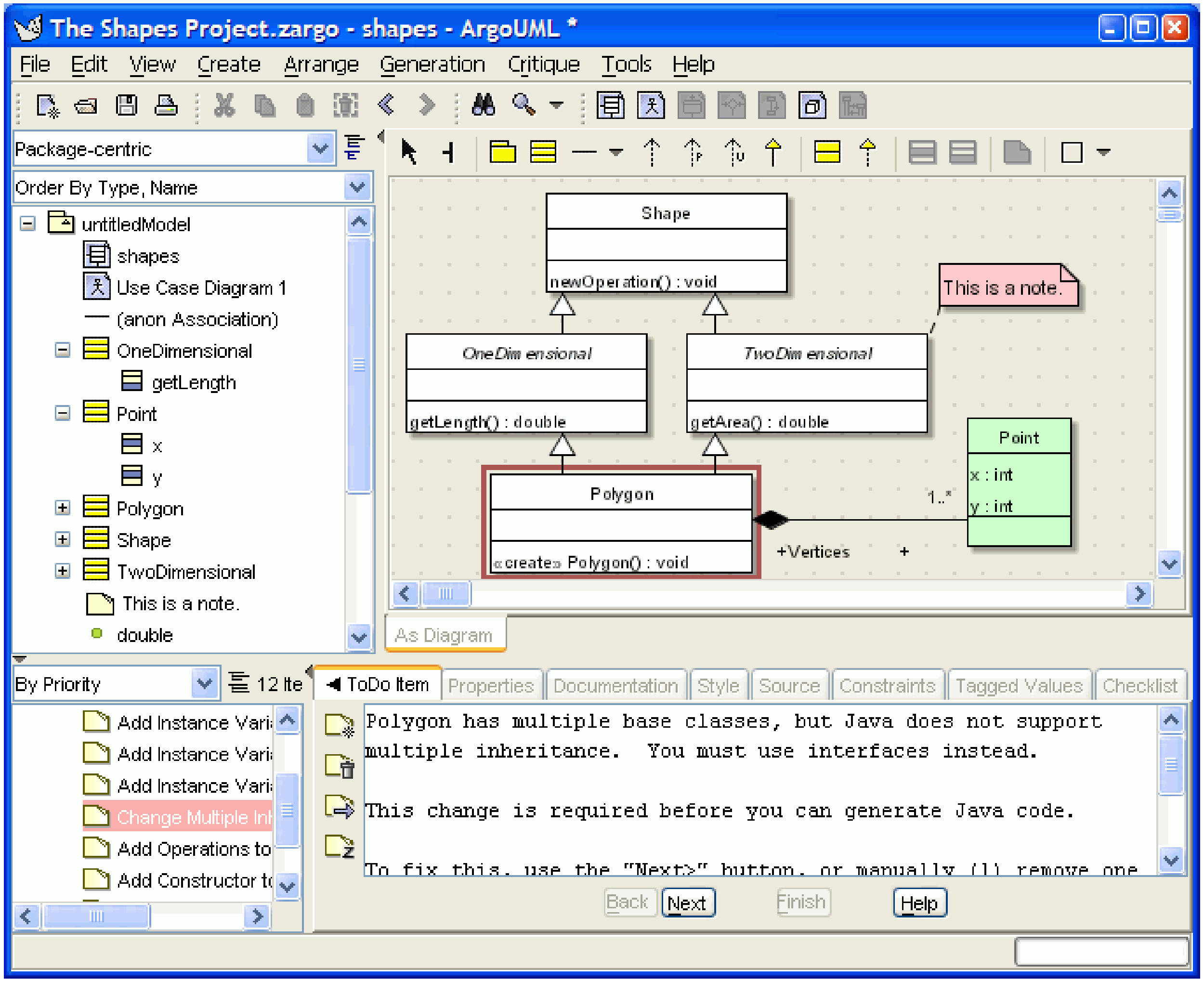
- Aims of this section:
    - Look at Argo, another software tool framework
    - Experience using Argo to develop a software tool from research prototype to near industrial strength tool

- Resources
    - ArgoUML website http://argouml.tigris.org/
        - Particularly Jason Robbin's PhD thesis and Tiziana Allegrini's dissertation
    - Cai, Y., Grundy, J.C. and Hosking, J.G. Experiences Integrating and Scaling a Performance Test Bed Generator with an Open Source CASE Tool, Proc 2004 IEEE Int Conf on Automated Software Eng, Linz, pp. 36-45.

Argo and ArgoUML

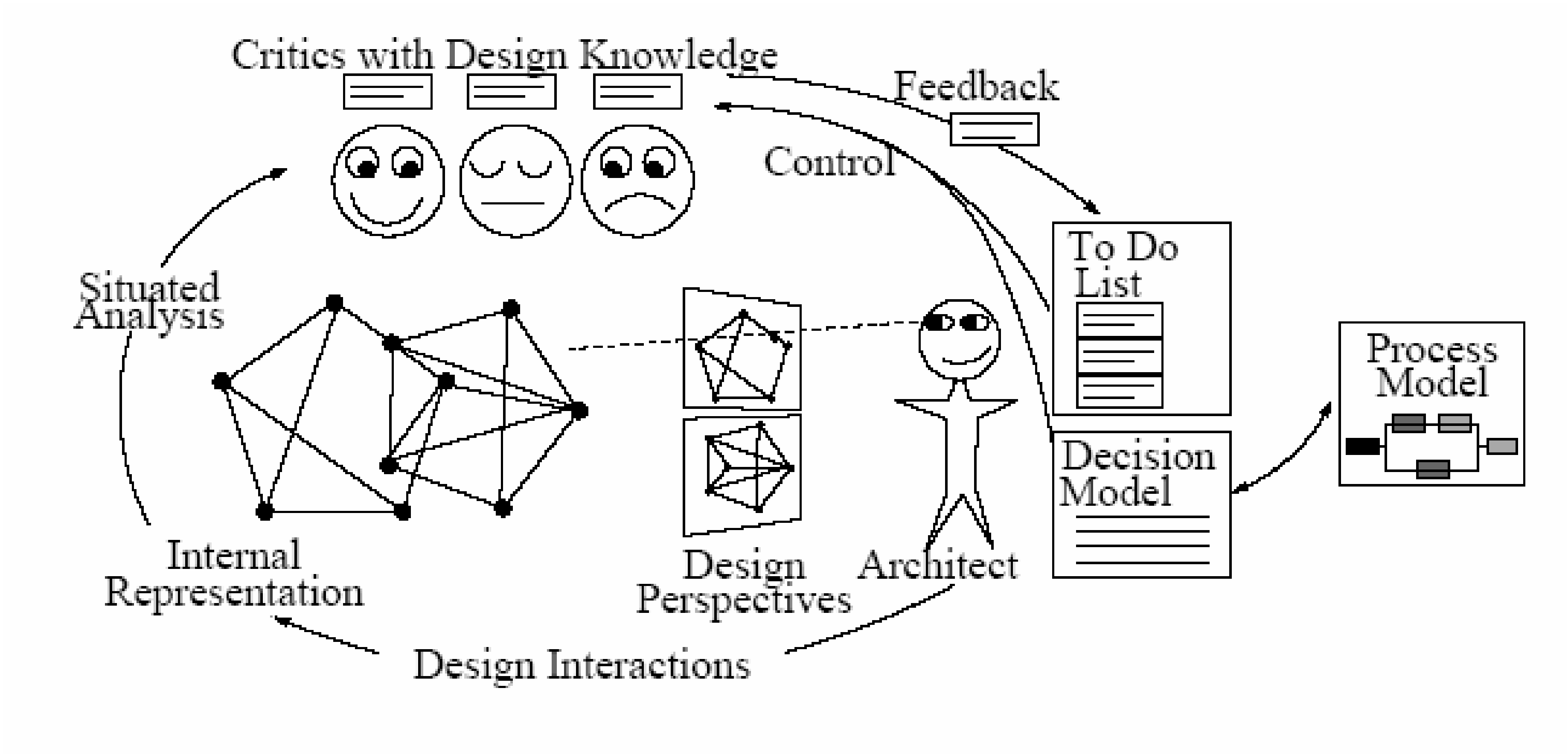
- Argo UML project goal: build an object oriented design tool that is:
    - a joy to use
    - actually helpful to designers when they are making design decisions, by offering cognitive support through critics
    - completely open source Java (FreeBSD license)
    - supporting everything in UML
    - modular and extensible
    - integrated with the web and other Tigris tools.
- Argo is the framework underneath the ArgoUML tool
- Strong influence on Eclipse and ArchStudio

ArgoUML in use



```
Polygon has multiple base classes, but Java does not support
multiple inheritance.  You must use interfaces instead.

This change is required before you can generate Java code.

To fix this, use the "Next>" button, or manually (1) remove one
```

Basic functionality

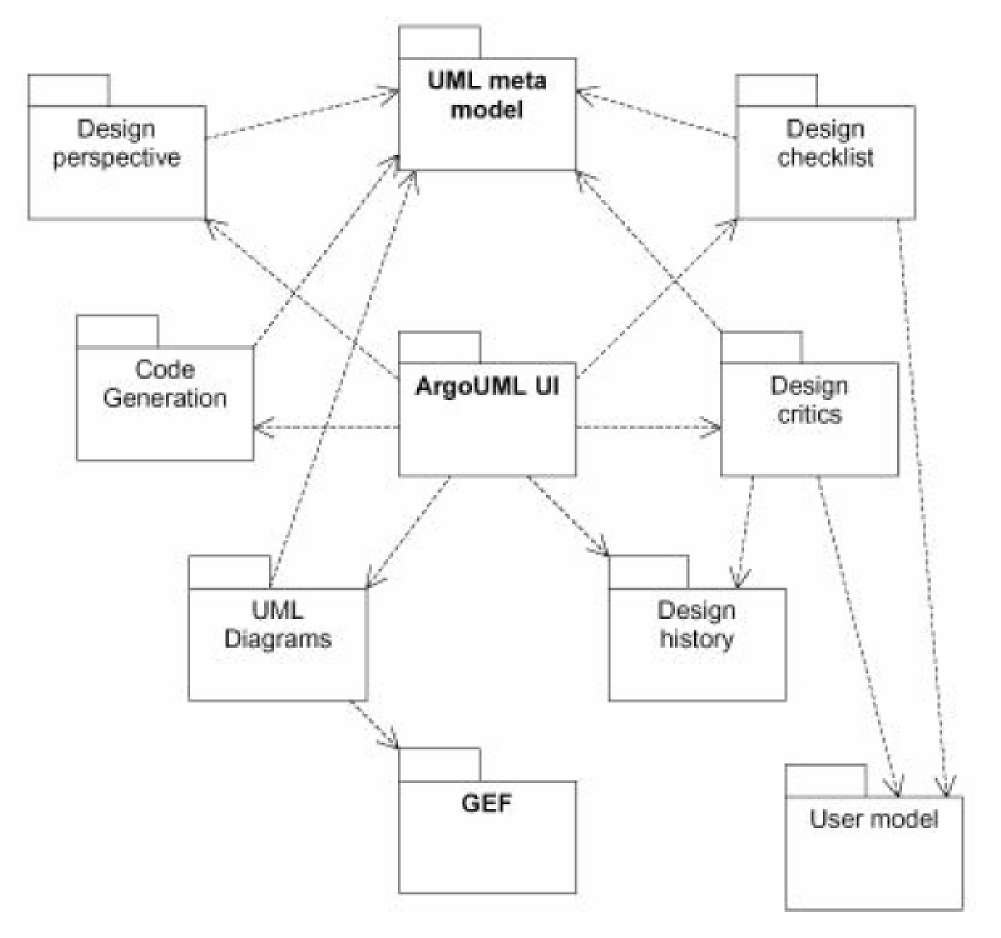


Basic functionality

- Design Perspectives
  - multiple views with consistency
- Critics
  - Multiple analysis tools which provide continual feedback on the design
- To do list
  - Feedback from critics presented here, provides active links to "criticised" design elements
- Process Model
  - Integrated process modelling using IDEF0 notation
  - Linked to critics, so can have task specific critics

Argo Architecture

- Major Packages:
- GEF
  - Graph Editing Framework provides reusable graph editing capabilities
- UML Meta Model
  - Based on NSUML open source UML meta model
- ArgoUML UI
  - Windowing and navigation
- Design Critics
  - Support for design critic implementation and predefined critics



Experience Applying Argo

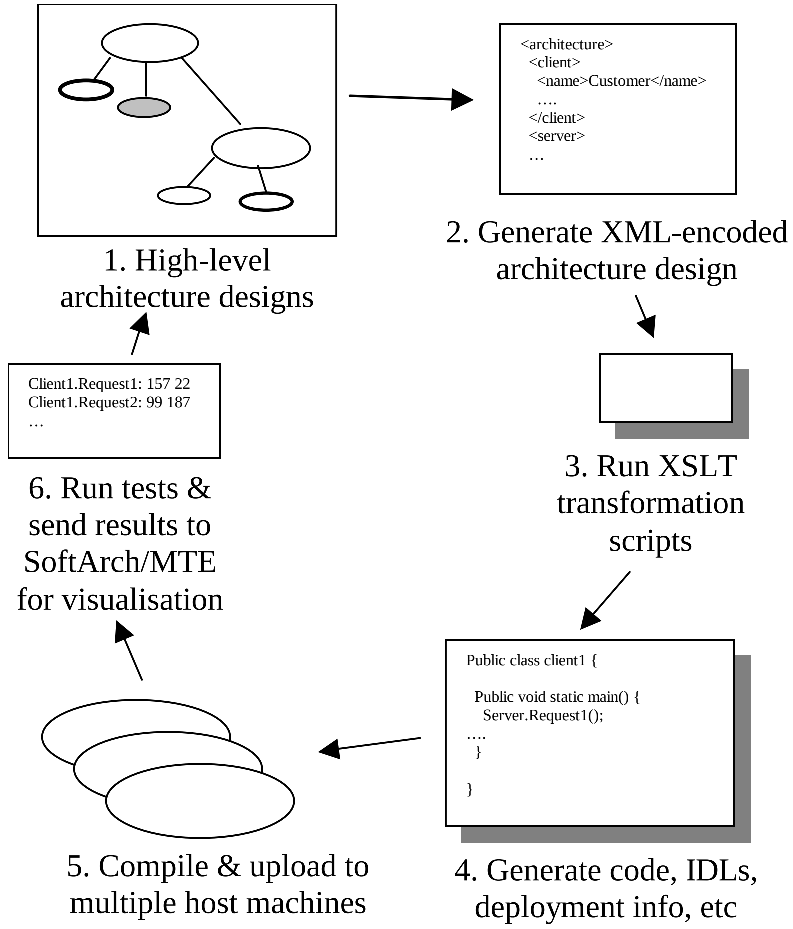
- SoftArch/MTE and its problems
- Re-engineered solution
- Experience
  - Integrating MTE with Argo/UML
  - XMI-derived model representation
  - Improvement of XSLT-based test bed generator
  - Using ANT
  - Result database
- Conclusions
  - Specific
  - Generalised

SoftArch/MTE

- SoftArch/MTE (ASE2001)

  - integrated environment to model and evaluate software architecture

  - automatically generates, compiles and deploys test bed code, runs performance tests, reports results



1. High-level architecture designs

```
<architecture>
  <client>
    <name>Customer</name>
    ….
  </client>
  <server>
  …
```

2. Generate XML-encoded architecture design

3. Run XSLT transformation scripts

```
Public class client1 {

  Public void static main() {
    Server.Request1();
  ….
  }

}
```

4. Generate code, IDLs, deployment info, etc

5. Compile & upload to multiple host machines

```
Client1.Request1: 157 22
Client1.Request2: 99 187
…
```

6. Run tests & send results to SoftArch/MTE for visualisation

SoftArch/MTE problems

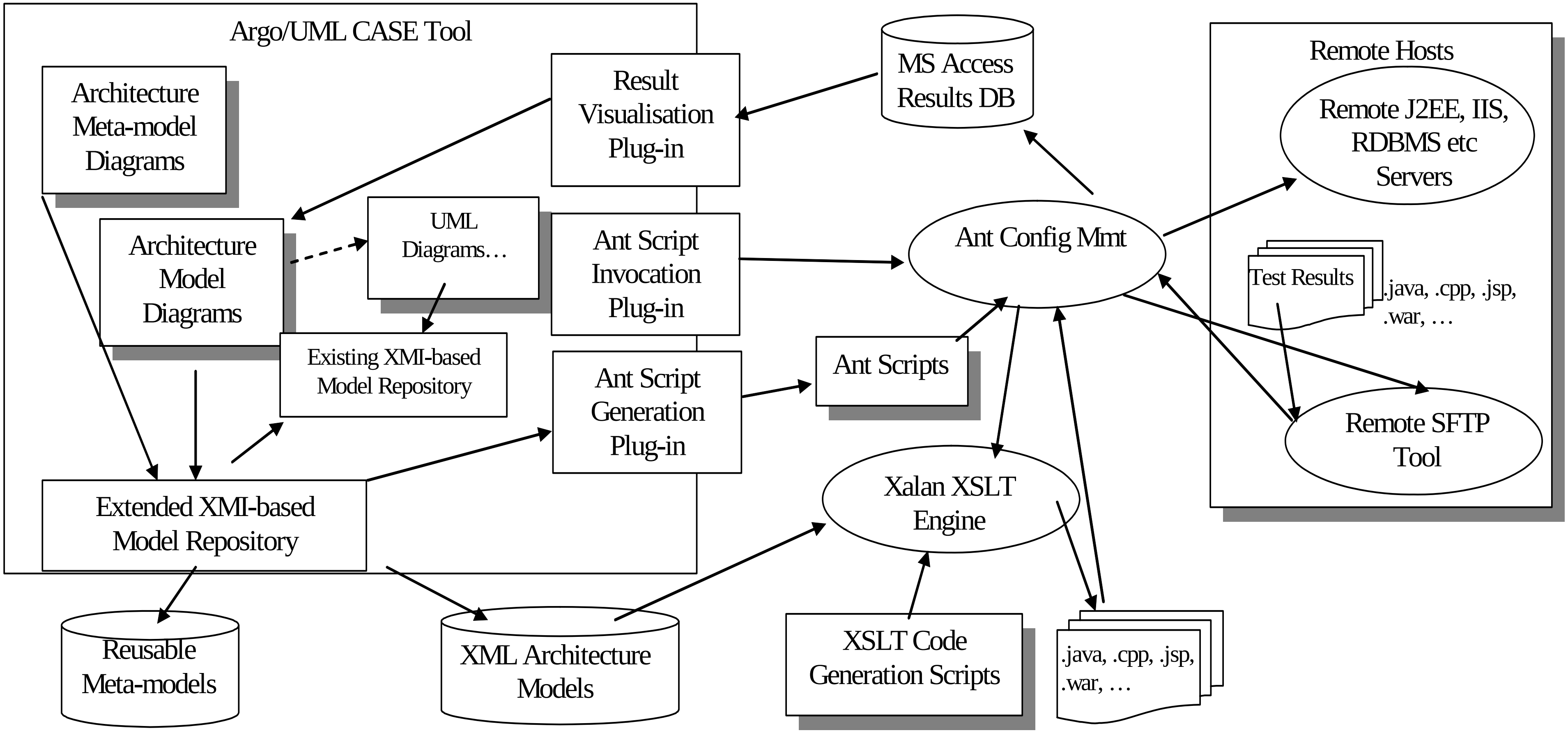
Problems when we applied SoftArch/MTE to several industrial case studies:

- custom framework (JViews)
- custom architecture notation
- custom XML representation
- non scalability of code generation approach
- custom deployment tool
- custom visualisation

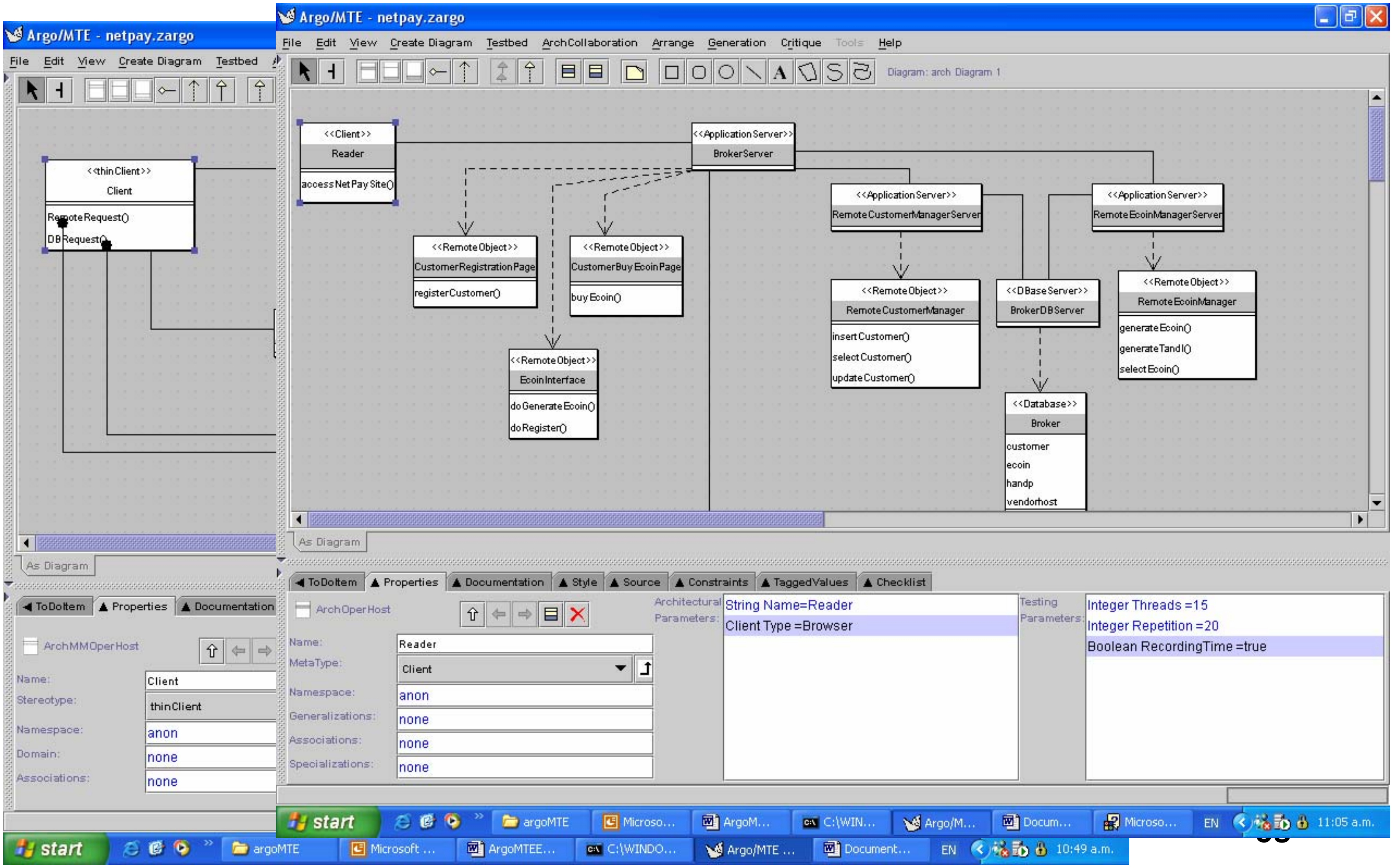
Re-engineered Solution

- Use Argo/UML as base tool
    - wider user base and more robust framework
    - integration with a standard UML modelling tool
- Extend UML meta model with arch descpn/perf elements
    - base on a more standard formalism
- Develop arch perf meta model and instance modelling tools in Argo
- Use standard XMI backend model representation
- Make XSLT based code generator more generic
- Use standard deployment tool (Ant)
    - Manages test code deployment and test run
- Use standard DB (Access) for result mmt and visuln

Re-engineered Solution


Argo/UML CASE Tool
Architecture Meta-model Diagrams
Architecture Model Diagrams
UML Diagrams…
Existing XMI-based Model Repository
Extended XMI-based Model Repository
Result Visualisation Plug-in
Ant Script Invocation Plug-in
Ant Script Generation Plug-in
MS Access Results DB
Ant Config Mmt
Ant Scripts
Xalan XSLT Engine
XSLT Code Generation Scripts
.java, .cpp, .jsp, .war, …
Reusable Meta-models
XML Architecture Models
Remote Hosts
Remote J2EE, IIS, RDBMS etc Servers
Test Results
.java, .cpp, .jsp, .war, …
Remote SFTP Tool

Argo/MTE Modelling



ArchOperHost

Name: Reader

MetaType: Client

Namespace: anon

Generalizations: none

Associations: none

Specializations: none

Architectural Parameters:
String Name=Reader
Client Type =Browser

Testing Parameters:
Integer Threads =15
Integer Repetition =20
Boolean RecordingTime =true

ArchMMOperHost

Name: Client

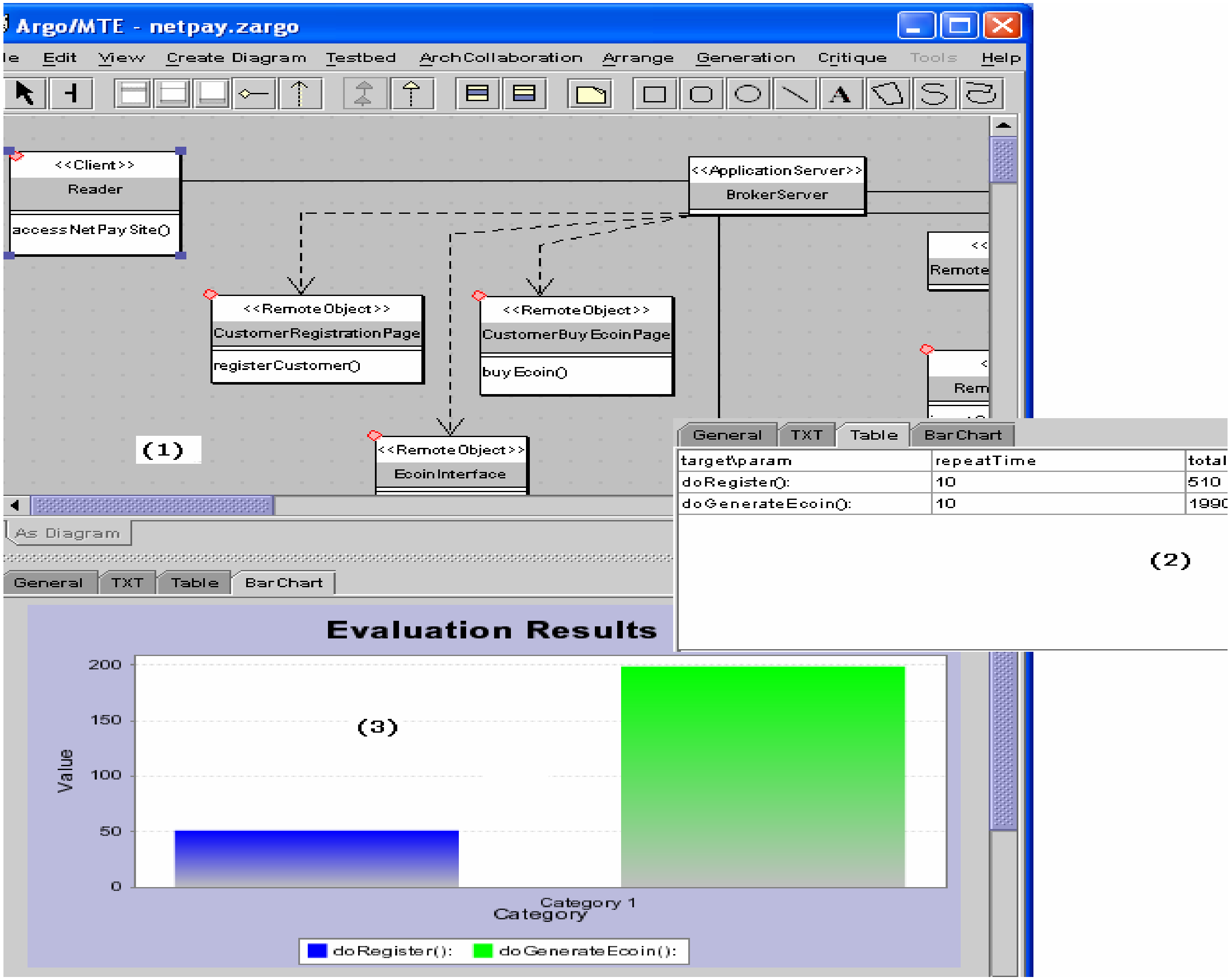
Stereotype: thinClient

Namespace: anon

Domain: none

Associations: none

Results visualisation



target\param	repeatTime	total
doRegister():	10	510
doGenerateEcoin():	10	1990

Conclusions

- Integrated Modelling Support

  Argo/MTE integrated with a standard UML-based CASE tool

  Allows test bed modelling and generation as a natural adjunct to UML modelling

  Reuses users' design notation knowledge reducing learning curve

  More appealing and effective environment than stand-alone SoftArch/MTE

- Enhanced data exchange capability

  Extended XMI model representation and extensible architecture meta-models increase chance of future model data exchange

Conclusions (cont'd)

- Better abstraction led to simpler code generation

  Addition of stereotype abstraction layer led to better reuse of code generation code & scripts

  Avoided the need for manual modification of code generation scripts

- Use of third-party tools

  Third-party tools used to coordinate:
  - test bed generation and execution process (Ant),
  - deployment (SFTP),
  - web-based client tests (ACT)
  - results management (Access)

  Much more scalable and flexible than our previous ad-hoc applications to perform these tasks.

  Particularly so for heterogeneous architectures incorporating several technologies

Generalised Conclusions

- Leverage third party tools in specialised domains
  - Complex dependency management
  - Scripting
  - Databases
  - Modelling tool implementation

  Avoid bespoke code (concentrate on your own strengths)
- Design for extendibility/reuse
  - Use abstractions to enhance reuse
  - Use plugin/API technologies to make integration easy
- Use standard representations where possible
  - Enhances user adoption
  - Enhances reuse and tool integration