

A Tour of the Eclipse Environment

Stephen Barrett

Concordia University – 6-Mar-2005



What is Eclipse?

Depends on who you ask.

What is Eclipse?

Man on the Street:

- Java-based development framework
- Spawned from IBM Visual Age/Smalltalk and Java, 1999 – initially 40 full-time developers
- Transitioned into open source project, 2001 – with IBM code base and funding
- Commercially viable, e.g., IBM WebSphere Studio, and SAP NetWeaver – both are extensions of Eclipse

What is Eclipse?

Application Programmer:

- IDE for Java, C/C++, Perl, Python, ...
 - Editor with cross-linked views
 - Code formatting and insertion
 - Code refactoring
 - State of art debugging
 - Unit testing
 - Build facilities

“An open extensible IDE for anything and nothing in particular.”
– eclipse.org

What is Eclipse?

Tool Builder:

- Extensible development platform
 - Collection of plug-in components that integrate seamlessly with the Eclipse environment.
 - Java Development Tools (JDT)
 - Plug-in Development Environment (PDE)
 - Graphical Editing Framework (GEF)
 - A framework and a set of services for building a development environment.
 - Maintained with an update manager plug-in.

What is Eclipse?

Architect / Designer:

- Implementation of model driven architecture (MDA) framework
- Modeling tool (textual and graphical)
- Data integrator, and model manager
- Sophisticated code generator
 - Observer, factory, proxy, etc. design patterns
 - Serialization for object brokering and persistence

Agenda

- IDE – Java coding with Eclipse
 - workbench highlights
- PDE – plug-in tool integration
 - packaging and management
- Philosophy – everything is a contribution
 - contribution rules
- Framework – making it all work
 - platform architecture
 - design principles and patterns
- EMF – the modeling framework (next week)
 - model unification

The IDE: Coding with Eclipse



Coding – The Java IDE

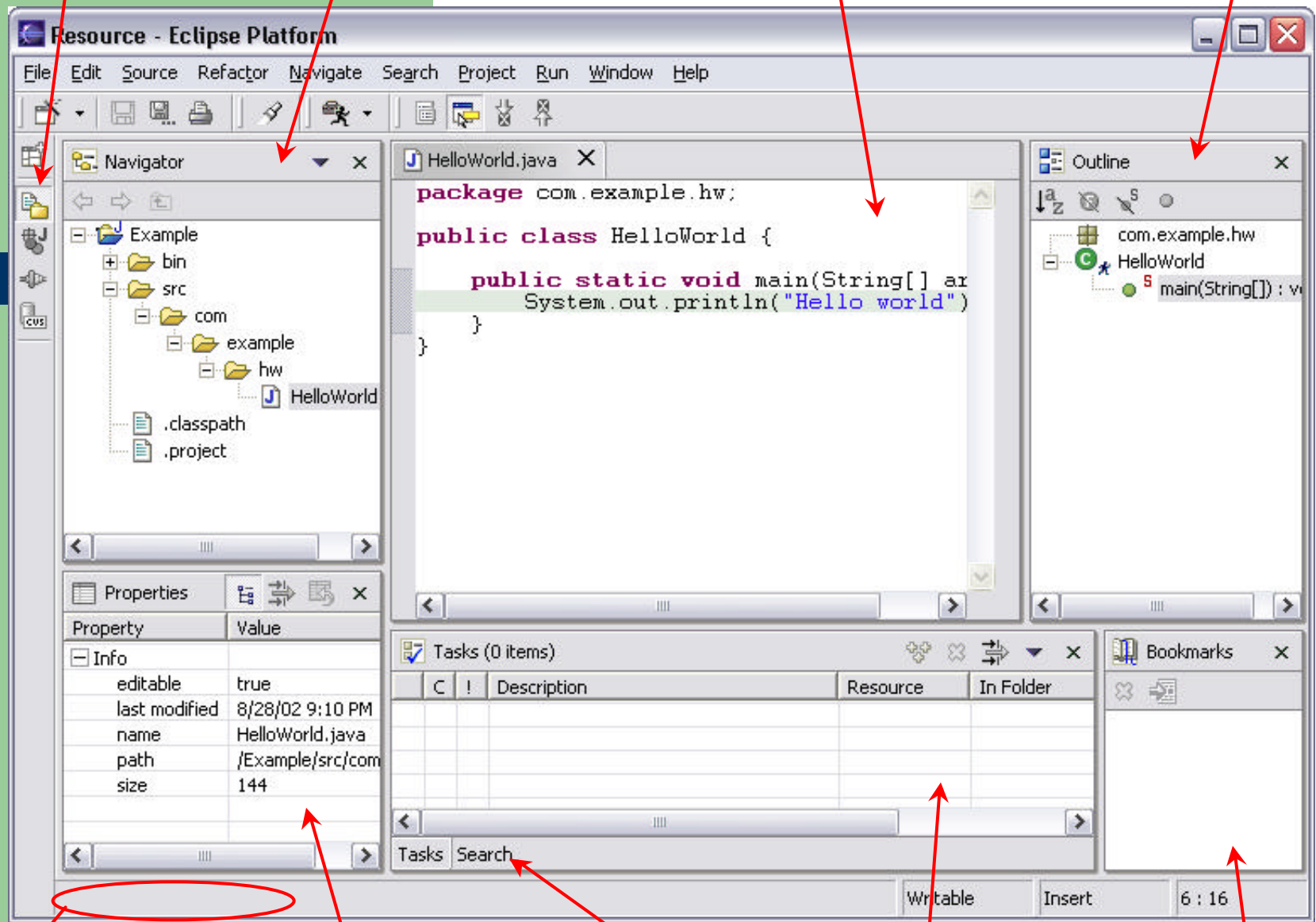
- Views, editors, and perspectives
 - View and editor linking
- Incremental compilation
- Full-featured debugger
- Configuration management (PVCS)
 - Quick Diff (requires baseline)

Perspectives

Resource navigator

Text editor

Outliner



Message area

Properties view

Stacked views

Tasks view

Bookmarks view

Coding – The Java IDE (cont'd)

- Javadoc insertion, formatting, and generation
 - GUI wrapper for javadoc command line tool
- Template-based code insertion
 - Define file bodies – code, comments, & Javadoc
- Code formatting; with preview
 - Set style for block constructs
 - Can format or restructure entire file
- Refactoring; with preview

Coding – The Java IDE (cont'd)

- Margin cues (errors, changes, tasks, etc.)
- Text folding
- Error highlighting (source, Quick Fix)
- Quick Fix (suggestions: Ctrl+1, click bulb)
- Quick Assist (helpful hints: Ctrl+1)
- Content Assist (methods, variables, source code, Javadoc, spelling, dialogs: Ctrl+Space)

IDE Walkthrough



The PDE: Creating Eclipse Tools



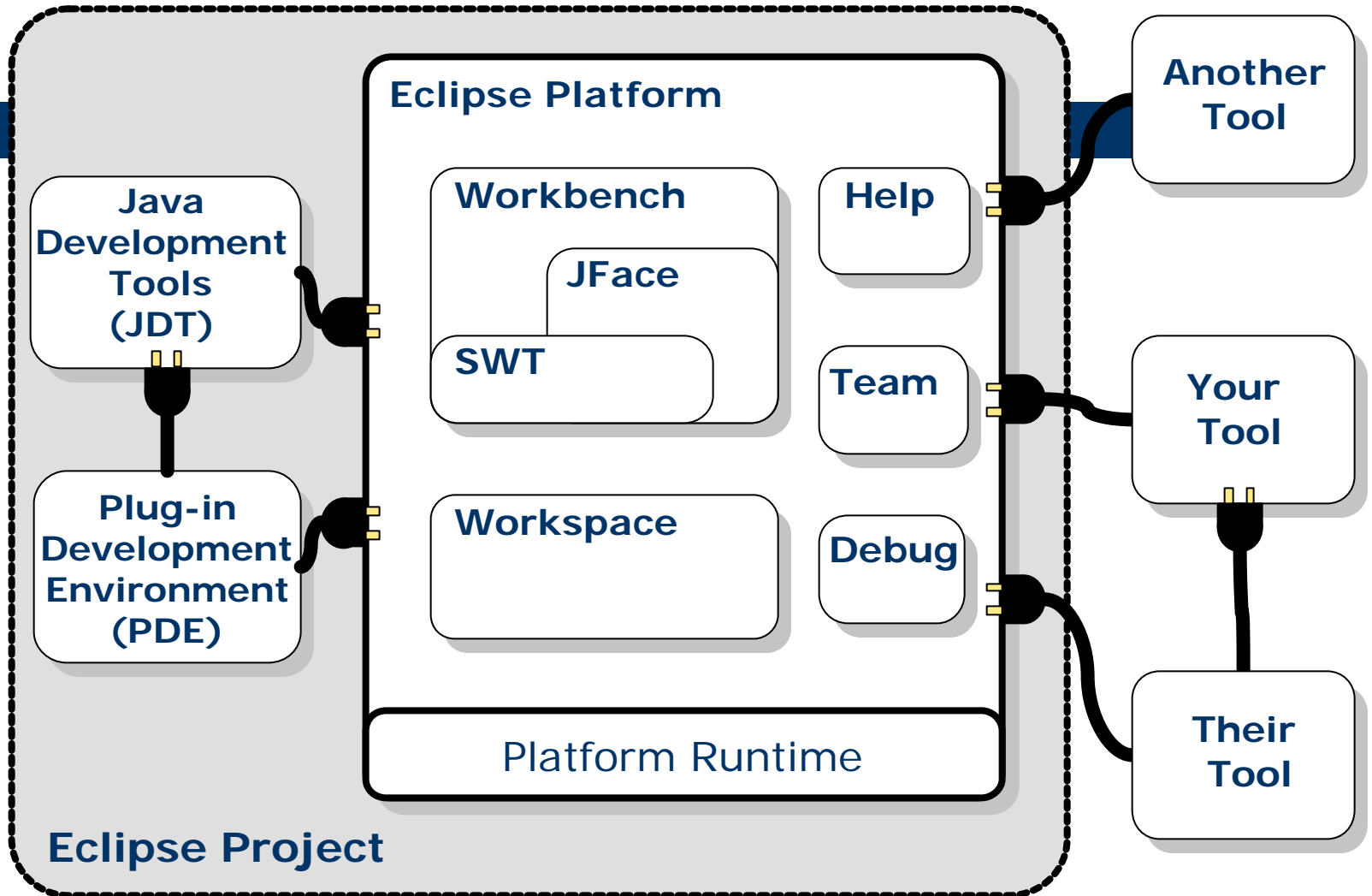
“...Emacs for the 21st century.”
– Martin Fowler

Tooling – Plug-ins

Extensions to the Eclipse framework are achieved through the use of plug-ins.

- Plug-ins are the basic unit of functionality – a component.
- Features are constructed from plug-ins.
- Both the Eclipse platform and its tools are composed of plug-ins.

Tooling – Plug-ins (cont'd)



Tooling – Plug-in Payoff

What does the framework buy us?

- Ease of tool discovery, installation, and removal – managed configurations
- Reuse of pre-developed functionality – don't reinvent the wheel
- Freedom of choice for the user
 - Choose the best solution.
 - Say NO to profit-driven feature proliferation!
- Reflective generation of viewers and editors

Tooling – Plug-ins in Action

Each plug-in includes or indicates everything needed to run it:

- *plugin.xml* – plug-in dependencies
- *feature.xml* – installed packages

On startup the Eclipse platform discovers all available plug-ins. (Not on my system!)

A plug-in is activated only when its code is actually needed.

Tooling – Plug-in Packaging

feature.xml describes and licenses the feature.

Lists included plug-ins and websites.

- URL – update and discovery sites.
- Requires – use of other plug-in extension points (contributions used)
- Plugin – plug-ins that comprise the feature.

Tooling – Plug-in Packaging

plugin.xml declares the interconnections to other plug-ins:

- Requires – dependencies on other plug-ins
- Extension-point – functionality available to other plug-ins (for collecting contributions)
- Extension – use of other plug-in extension points (contributions used)
- Exports – class visibility to other plug-ins

Plug-in Walkthrough

Required for Assignment 3 (release builds):

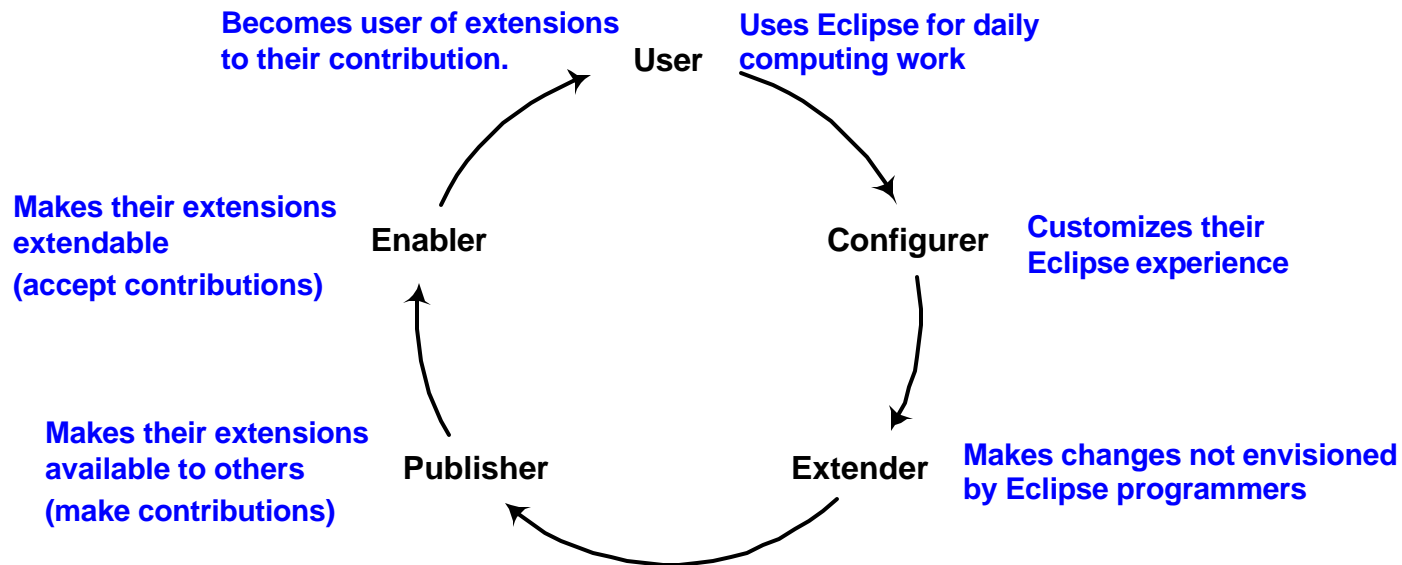
- Java 2, Standard Edition 1.4.2 or greater
 - <http://java.sun.com/j2se/>
- Eclipse 3.0.1 (platform, JDT, PDE, UI, etc.)
 - <http://eclipse.org/downloads/>
- Eclipse Modeling Framework 2.0.1 (EMF, SDO, etc.)
 - <http://eclipse.org/downloads/>
- Graphical Editor Framework 3.0.1 (GEF)
 - <http://eclipse.org/downloads/>
- UML2 1.0.1
 - <http://eclipse.org/downloads/>
- Omondo EclipseUML 2.0.0
 - <http://www.omondo.com/download/free/>

The Rules: Contributing to Eclipse



Contributing – User as Programmer

Empower the user in a learning environment



“Hell, there are no rules here – we're trying to accomplish something.”
– Thomas Edison

Contributing – Rules

Eclipse has a strong world view, leading to two dozen rules. To play you must play by the rules!

- Contribution Rule
 - Everything is a contribution.
- Lazy Loading Rule
 - Contributions are only loaded when they are needed.
- Sharing Rule
 - Add, don't replace.
- Conformance Rule
 - Contributions must conform to expected interfaces.
- Monkey See / Monkey Do Rule
 - Always start by copying the structure of a similar plug-in.

“There are three rules for writing the novel.
Unfortunately, no one knows what they are.”
– W. Somerset Maugham

Contributing – Rules (cont’d)

- **Relevance Rule**
 - Contribute only when you can successfully operate.
- **Safe Platform Rule**
 - Extension point providers, must protect themselves against misbehavior on the part of extenders.
- **Invitation Rule**
 - Whenever possible, let others contribute to your contributions.
- **Fair Play Rule**
 - All clients play by the same rules, even me.

“There are two rules for success:
1) Never tell everything you know.”
– Roger H. Lincoln

Contributing – Rules (cont’d)

- **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.
- **License Rule**
 - Always supply a license with every contribution.
- **Program to API Contract Rule**
 - Check and program to the Eclipse API contract.

“The rule is, jam tomorrow and jam
yesterday—but never jam today.”
– The White Queen

Contributing – Rules (cont’d)

- Integration Rule
 - Integrate, don’t separate.
- Responsibility Rule
 - Clearly identify your plug-in as the source of problems.
- User Arbitration Rule
 - Let the user decide which contribution to use.
- Other Rule
 - Contributions that don’t typically apply to the current perspective appear in an **Other...** dialog.

“All human rules are more or less idiotic.”
– Mark Twain

Contributing – Rules (cont’d)

- **Explicit API Rule**
 - Separate the API from the internals.
- **Stability Rule**
 - Once you invite others to contribute, don’t change the rules.
- **Defensive API Rule**
 - Reveal only the API in which you have confidence, but be prepared to reveal more.
- **User Continuity Rule**
 - Preserve the user interface state across sessions.

“Rules? We don’t need no stinkin’ rules!”
– The Treasure of the Sierra Madre

Contributing – Rules (cont’d)

- Adapt to `IResource` Rule
 - Whenever possible, define an `IResource` adapter for you domain objects.
- Strata Rule
 - Separate language-neutral functionality from language-specific functionality and separate core functionality from UI functionality.

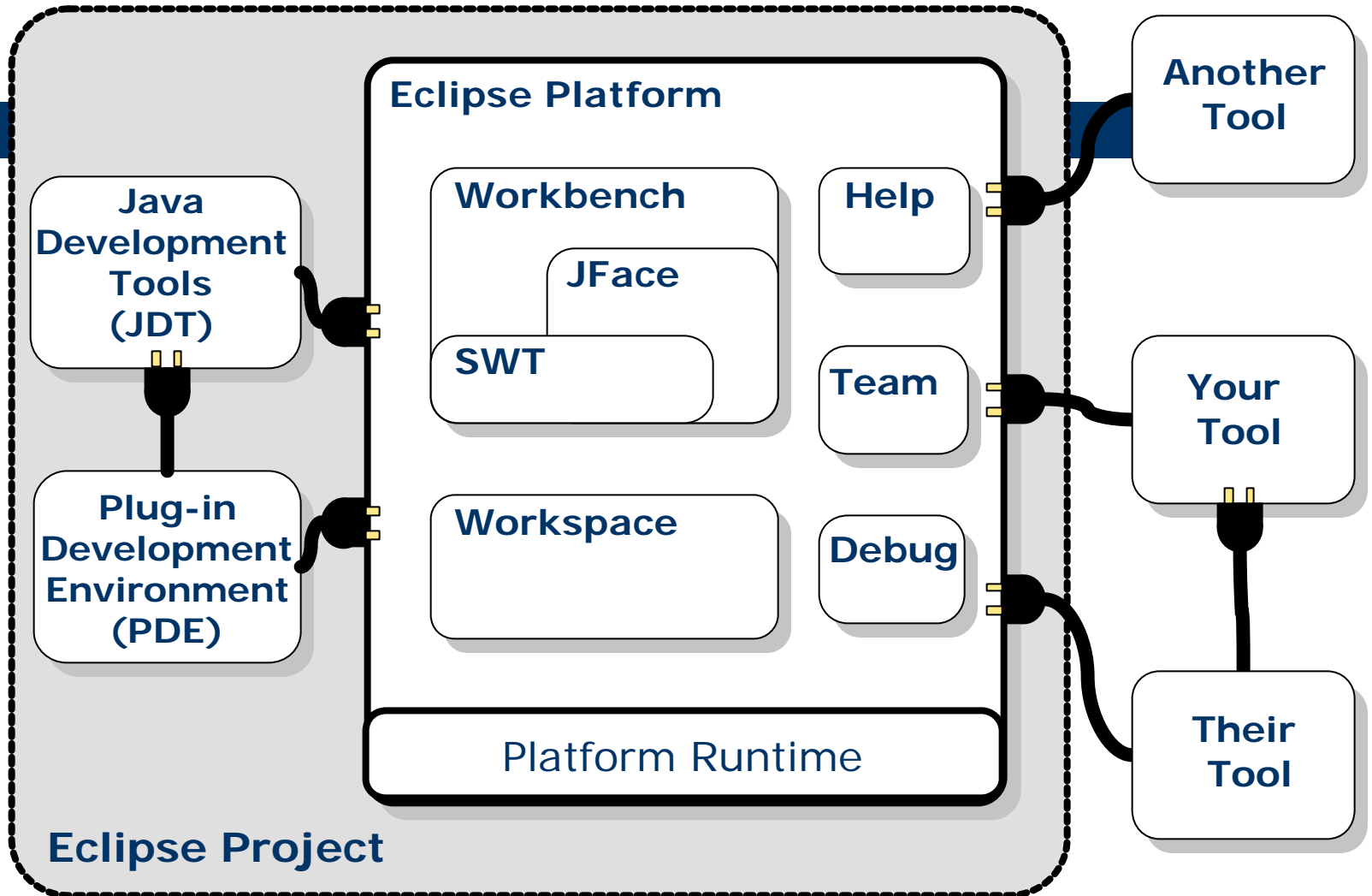
The Framework: Building Eclipse

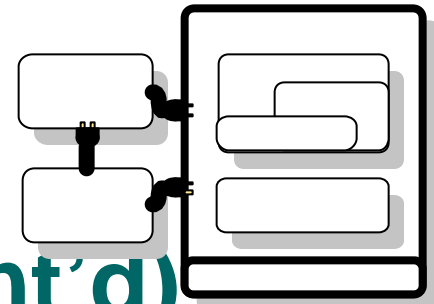


Building – Architecture Goals

- Define a common programming-language-neutral infrastructure
- Allow tools to interoperate
 - Often in ways not imagined by the tool writers.
- Deliver power to the user
 - Who will play by the rules!

Building – Architecture





Building – Architecture (cont'd)

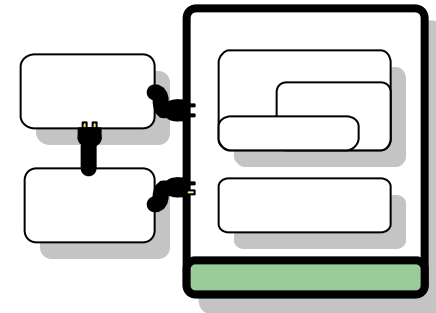
- Platform – common programming-language-neutral infrastructure
- JDT – adds full-featured IDE to Eclipse
- PDE – extends JDT for plug-in development
- Runtime – discovers plug-ins on start-up and manages plug-in loading
- Workspace – manages one or more top-level projects
- SWT – Provides graphics and defines standard set of widgets
- JFace – common UI tasks built on top of SWT
- Workbench – defines the Eclipse UI paradigm.

Building – Design Patterns

Motivation:

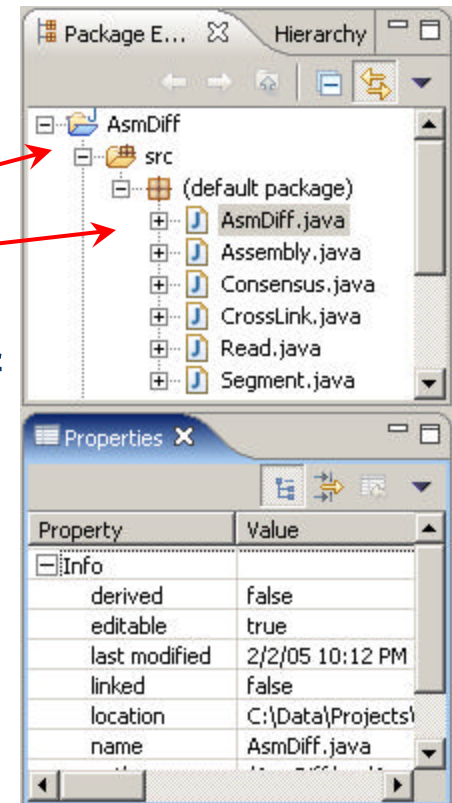
- Principles of design underlying Eclipse structure are also behind platforms of all kinds.
- Demonstrate how patterns play out in various contexts.
- Learn Eclipse design to more quickly learn how to play according to the Eclipse rules.
 - Rules are really patterns in a micro format.
 - Gamma and Beck claim that six hours of reading Eclipse code and only one hour of typing code is productive.

Core Runtime – Extensions

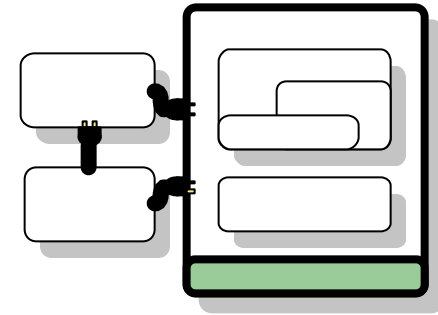


Eclipse strictly separates UI parts from non-UI parts:

- plug-in `org.eclipse.core.resources`
 - Defines interfaces `IFolder`, `IFile`, etc.
- How does Properties view get details of selected object in Package Explorer?
 - View needs an interface from which to fetch properties.
 - File object needs to support this interface.



Core Runtime – Extensions



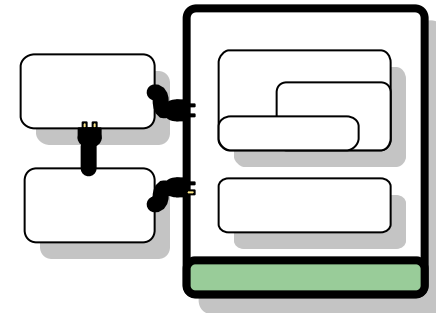
Interface IPropertySource

```
public interface IPropertySource
{
    public Object getEditableValue();
    public IPropertyDescriptor[] getPropertyDescriptors();
    public Object getPropertyValue(Object id);
    public boolean isPropertySet(Object id);
    public void resetPropertyValue(Object id);
    public void setPropertyValue(Object id, Object value);
}
```

How does IFile surface this interface?

- `public interface IFile extends IPropertySource`
 - bloats class interface
 - breaks public API
 - couples model to UI

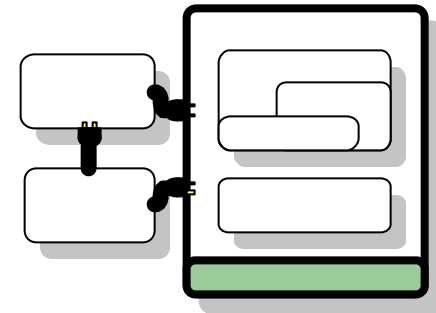
Core Runtime – IAdaptable



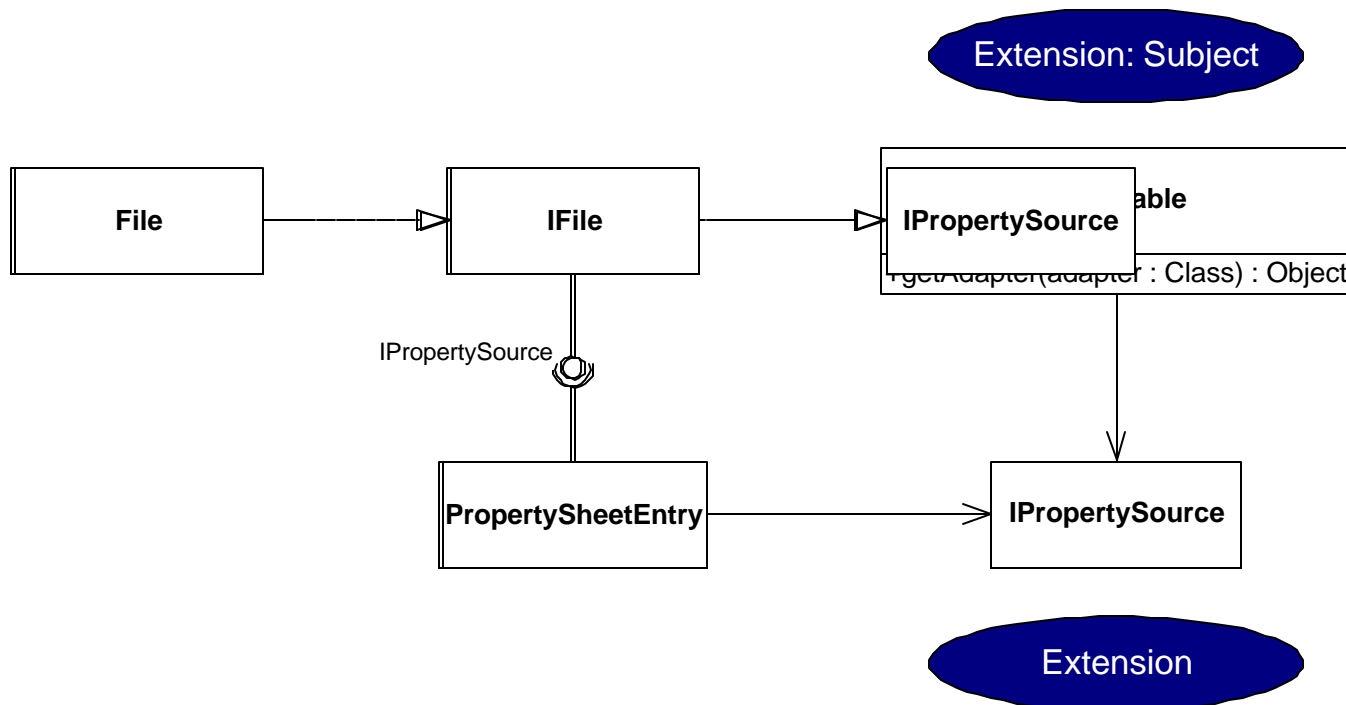
Use Extension Object pattern, allows us to:

- Add a service interface to a type without exposing it in the type (keep API contract).
- Add behavior to preexisting types like IFile.
- Let clients query whether an object has a particular extension.

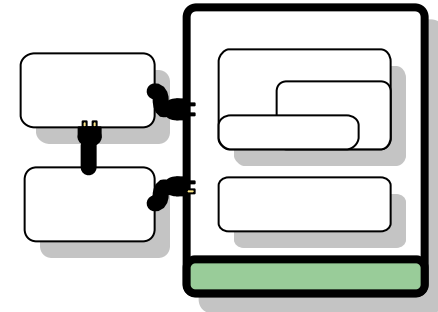
Core Runtime – IAdaptable



What we want (sort of):



Core Runtime – IAdaptable



Interface IAdaptable

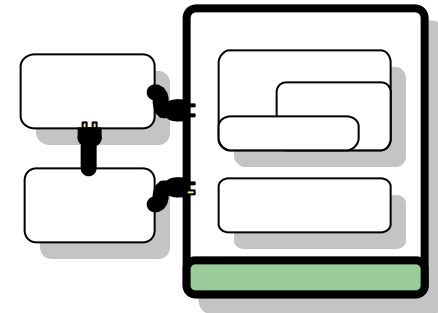
```
public interface IAdaptable
{
    public Object getAdapter(Class adapter);
}
```

Class PropertySheetEntry

```
private IPropertySource getPropertySource(Object object)
{
    //...
    else if (object instanceof IAdaptable)
        result = (IPropertySource)((IAdaptable) object).
            getAdapter(IPropertySource.class);

    return result;
}
```

Core Runtime – IAdaptable



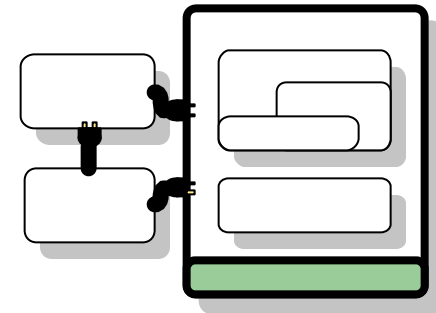
Class may implement `getAdapter()`

- Additional interfaces will alter implementation of `getAdapter()`, not the class API.

A factory may implement `getAdapter()`

- Requires no code changes to be made to the existing class.

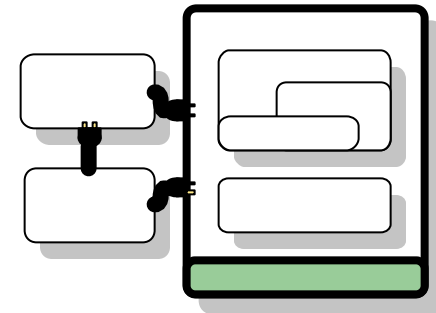
Core Runtime – IAdaptable



Adapter factory allows clients to use interface (IFile, say), and extend its behavior.

1. Implement an AdapterFactory with the adapters to be added to IFile.
2. Register factory with the platform's AdapterManager, a façade object.

Core Runtime – IAdaptable



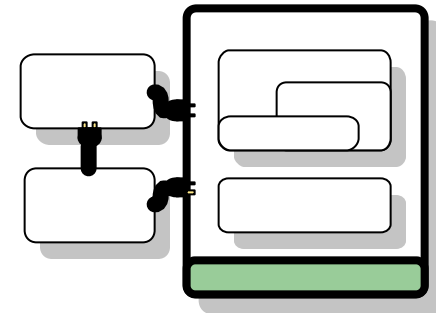
Factory declares adapters it provides:

```
Class FileAdapterFactory implements IAdapterFactory
{
    public Class[] getAdapterList()
    { return new Class[] { IPropertySource.class }; }
    //...
}
```

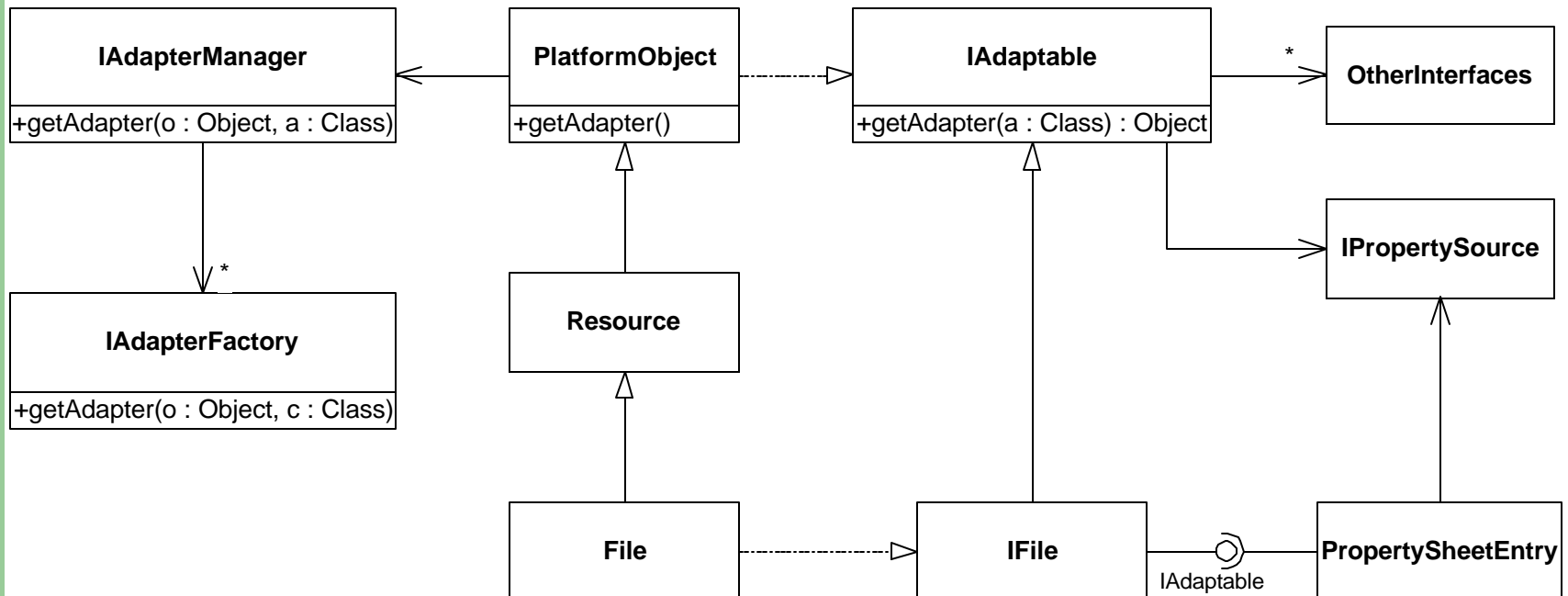
And implements getAdapter:

```
public Object getAdapter(Object o, Class adapter)
{ //...
    else if (adapter == IPropertySource.class)
        return new FilePropertySource((IFile) o);
    //...
}
```

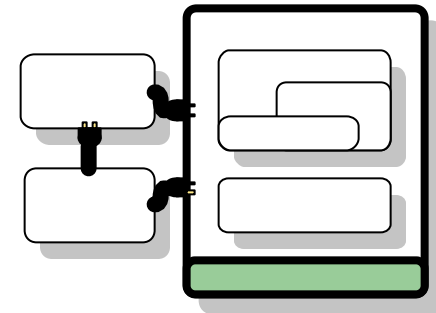
Core Runtime – IAdaptable



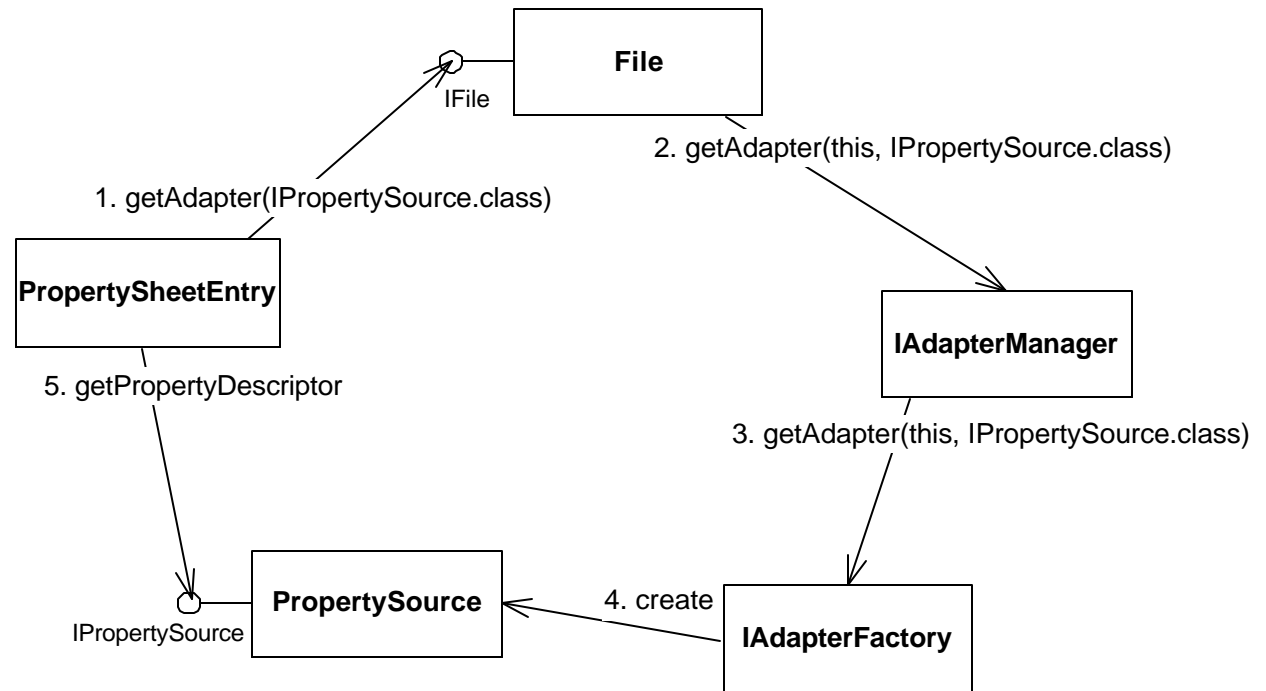
What we really get!



Core Runtime – IAdaptable



Object Collaborations



The End: Questions?



References

- Frank Budinsky et al, Eclipse Modeling Framework: A Developer's Guide, Addison-Wesley, 2004
- Eric Gamma and Kent Beck, Contributing to Eclipse: Principles, Patterns, and Plug-ins, Addison-Wesley, 2004
- Ed Merks, The Eclipse Modeling Framework: Introducing Modeling to the Java Technology Mainstream, JavaOne presentation, 2004
- Bill Moore et al, Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework, IBM Redbook, 2004
- www.eclipse.org