

# How to rapidly add language support into Eclipse.

## The SmartFrog Eclipse plug-in (a case study)

Elwin Ho  
Julio Guijarro  
Diana Friedland

© 2005 Hewlett-Packard Development Company, L.P.  
The information contained herein is subject to change without notice

# Agenda

- TBD

# What Eclipse plug-ins I have developed

- HP SmartFrog Eclipse plug-in (SmartFrog language, Java)
- HP OpenCall Media Platform vXML developer toolkit (VoiceXML, JSP and Java)
- HP OpenCall Media Platform OClet development environment (Java with new library)
- Xmen: XML editor for Eclipse(XML)
- HP Resource Explorer

# The challenge

To support a new language in Eclipse, you need:

- Language Editor: Text or graphic editor;
- Integrated online help
- Language debugger
- Deployment
- Leverage existing tools .....

To add to the challenge, consider the following:

- Have you implemented enough features?  
Complete coverage?
- Will you have enough time to implement?
- Which feature do you want to implement first? **JG1**
- What solutions are available?



JG1 You could use:  
Which feature do you want to implement first?/ Which feature is more useful for your users?  
Julio G, 1/26/2005

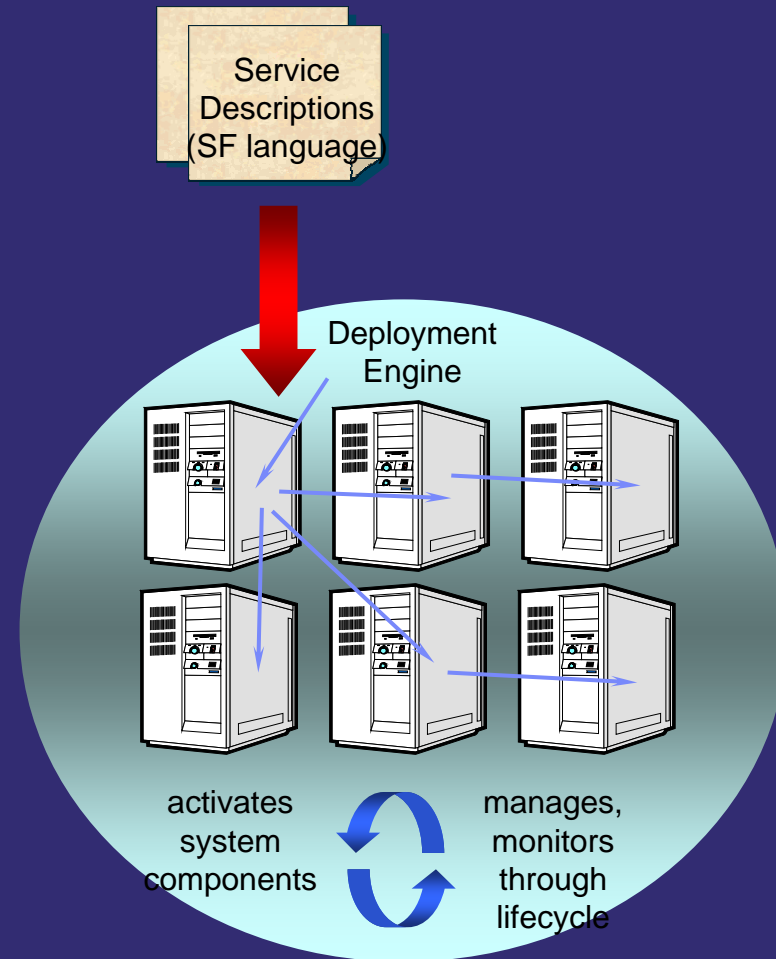
# Example: SmartFrog Eclipse plug-in project

SmartFrog is a technology for describing distributed software systems as collections of cooperating components, and managing them through the entire application lifecycle. ([www.smartfrog.org](http://www.smartfrog.org))

SmartFrog consists of a **language** for describing component collections and component configuration parameters, and a **runtime environment** which activates and manages the components to deliver and maintain running systems.

SmartFrog application development involves :

- develop SmartFrog description files. (SmartFrog language)
  - develop Java components.
  - use the RMI compiler to generate the SmartFrog executable files.
  - deploy, test, and debug the application in separate manual steps.
- 
- Have a fixed time frame, need to prioritize the problems.
  - Difficult to prioritize the steps, don't know what's available for each functionality.
  - Have some existing tools. leverage it or not.



# The solution

- What is enough?
  - List all important steps in each Software development cycle :Define, Design, Coding, Test, and Deploy. (And find out what's missing. )
- What solutions are available?
  - List out all levels of available solutions. e.g:
    - Document the manual steps
    - Use external component through
      - Add entry in “External tools” to invoke external component
      - Add entry in menu/tool bar, invoke external component
    - Implement the feature with native Eclipse API
- What to do first?
  - Depend on which process is rare, or recurring. (After you get the list of features you need, you can easily find out the priority.)
  - Implement the basic features that other features depend on first. (e.g. project creation first, then editor, so that you can extend it)
- Use Eclipse as a integration tool

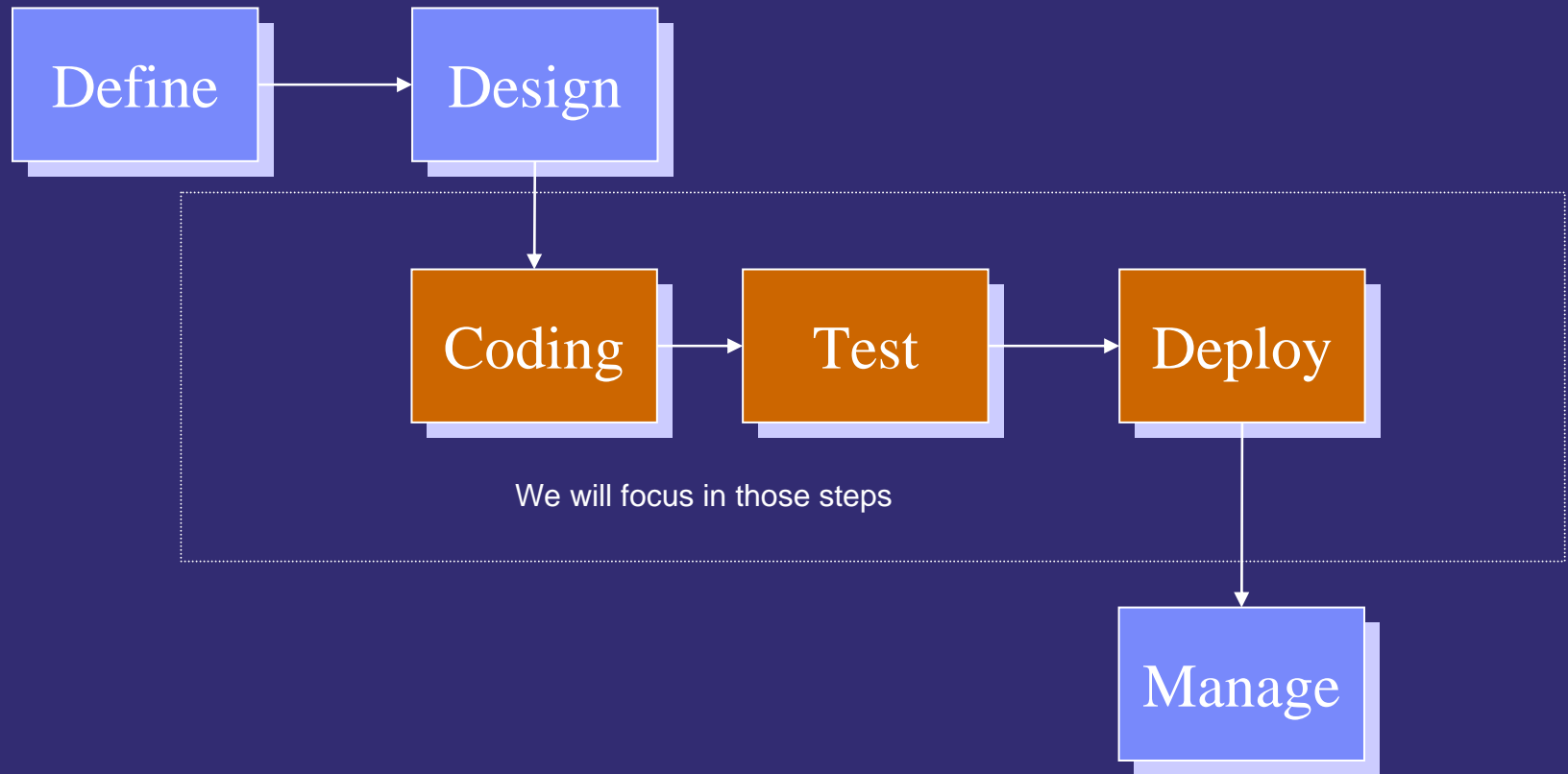
## Use SmartFrog project as the case study

JG3

I think you should clarify where Eclipse helps and where it does not. What is easy and what is not. Also try to conclude where new Eclipse tools/libs are needed.

Julio G, 1/26/2005

# Typical Software Development cycle



# Coding

## Features for coding step:

- Project creation
- File creation
- Editing
- Build/ Error detection
- Analyze
- Perspective/Preferences
- Online help
- Search
- Version control

# Coding: Project creation

What can you do?:

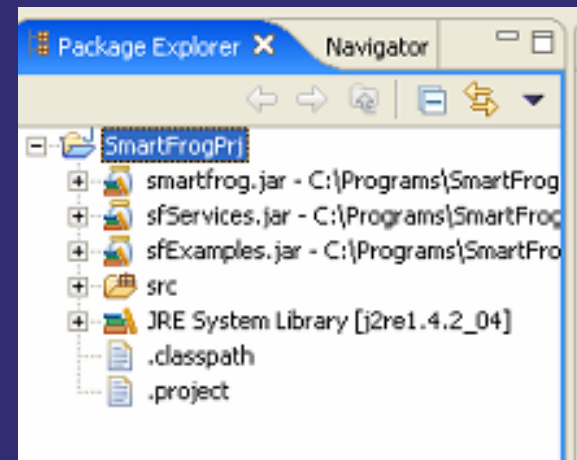
- Create a basic project structure.
- Associate your own builder for the project.
- Create template application.
- Add default libraries.

How do you do it?:

- Invoke external legacy tools through:
  - Run>Externals Tools
  - Add a Menu/Tool bar
- Eclipse project wizard.

Why we do that in SmartFrog?:

- Basic steps.
- Easy to implement.



# Coding:Files creation

What can you do?:

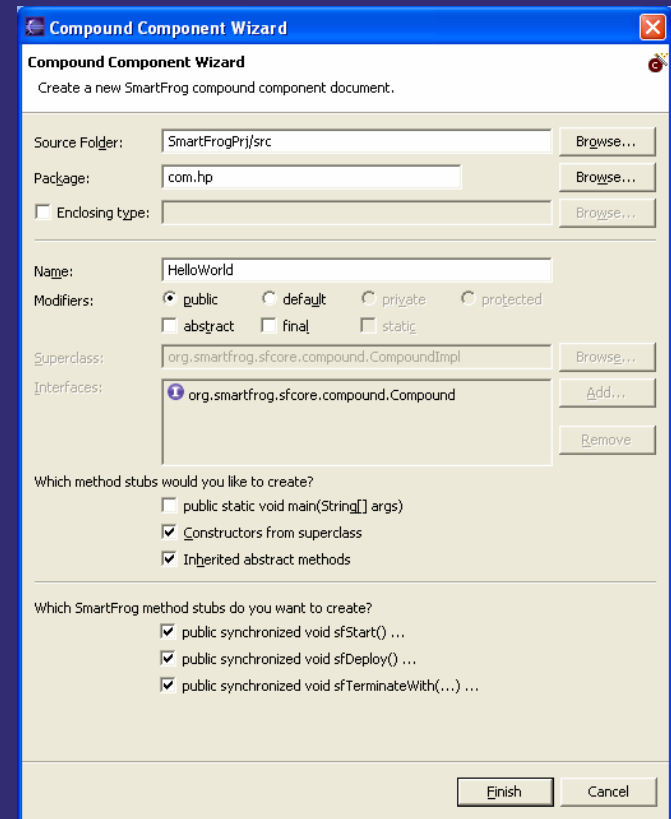
- Create sample components
- Allow user customize the skeleton/template

How do you do it?:

- Eclipse Wizard framework
- Use something Similar Java DOM/AST
- Eclipse XDT ..

Why we do that in SmartFrog?:

- Recurring feature
- Save unnecessary typing time
- Customer wants it



# Coding: Editing

## What can you do?:

JG2

- Associate the editor to the new file
- Support standard editing (copy/paste/printing)
- Syntax coloring
- Code completion
- Hovering help
- Marker .....

## How do you do it?:

- External editor (vi, emacs, remember refresh the project)
- Extends default text editor
- Extends Xmen for XML editor
- Use Eclipse editor framework

## Why we do that in SmartFrog?:

- Most recurring feature

```

hello.sf
#include "org/smartfrog/examples/helloworld/printer.sf"
#include "org/smartfrog/examples/helloworld/generator.sf"

sfConfig extends Compound {
  #include
  ATTRIB
  Compound
  HOST
  LAZY
  NULL
  PARENT
  PROPERTY
  Prim
  ROOT
  extends
  print to the instance of the printer
  */
  printer LAZY ATTRIB p;
}
/**
 * Printer component
 */
p extends Printer {
  name "myPrinter";
}
  
```

**JG2** A challenge: Sych java components and SF description. Wizard can help in generating skeleton for both  
Julio G, 1/26/2005

# Coding: Analyze

## What can you do?:

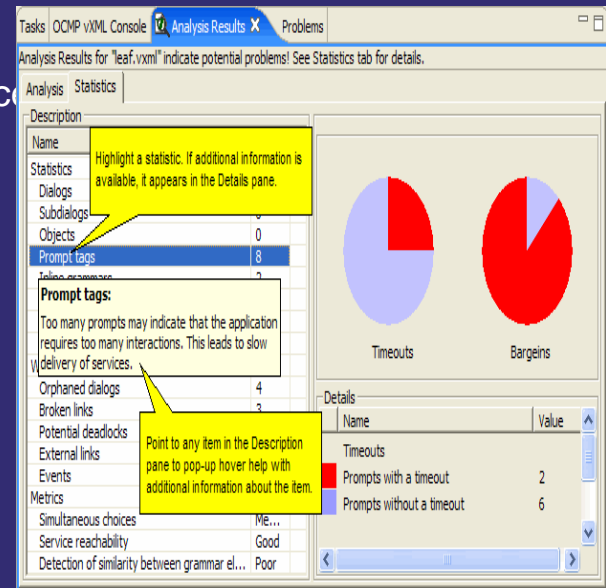
- Help user automatically detect potential problem
- Use task list to show results, and marker to link to source code that cause problem
- Correction proposal

## How do you do it?:

- Implement your analyzer and invoke externally
- Use task viewer and bookmarker. (pictures)
- Use AST to update the code
- Implement your own analyzer to help developer create better code, maybe integrate it into the builder

## Why we do that in SmartFrog?:

- Low priority
- Difficult to implement



(VoiceXML plug-in)

# Coding: Builder

## What can you do?:

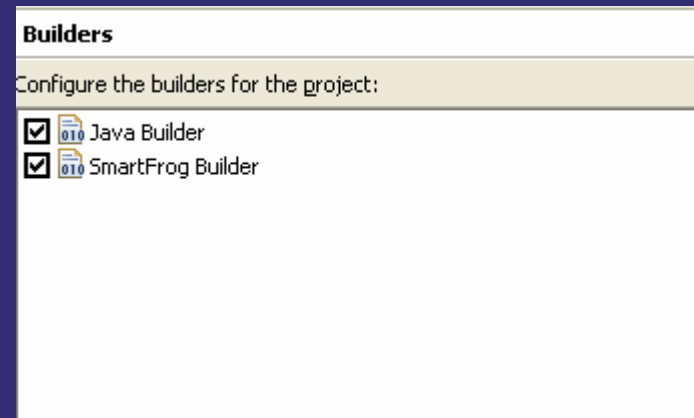
- Generate code from the application.
- Customize the build step. e.g., Using RMI
- Ant support

## How do you do it?:

- Implement your own builder
- Use build
- Use Ant Extension ??

## Why we do that in SmartFrog?:

- Complicated steps and easy to make mistake



# Coding: Perspective/Preference

## ■ Perspective

### What can you do?:

Define the initial look of your workbench window. Allow user to easily access the necessary tools.

### How do you do it?:

Implement IPerspectiveFactory

## ■ Preferences

### What can you do?:

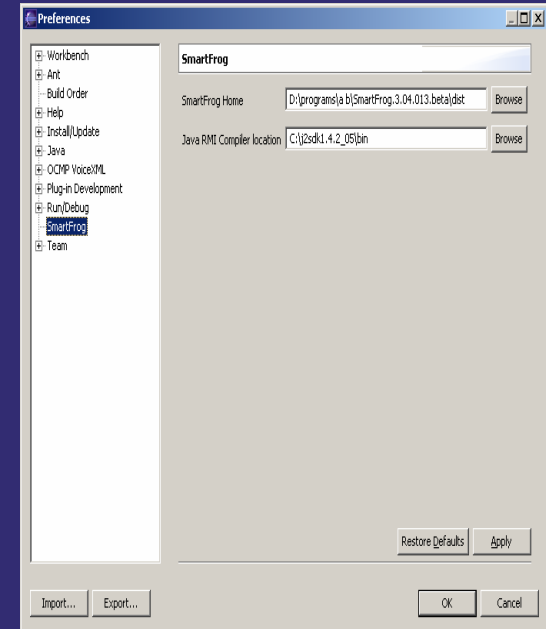
Allow user to define global or local setting

### How do you do it?:

Implement IWorkbenchPreferencePage

### Why we do that in SmartFrog?:

- Easy to implement
- Must have



# Online help integration

## What can you do?:

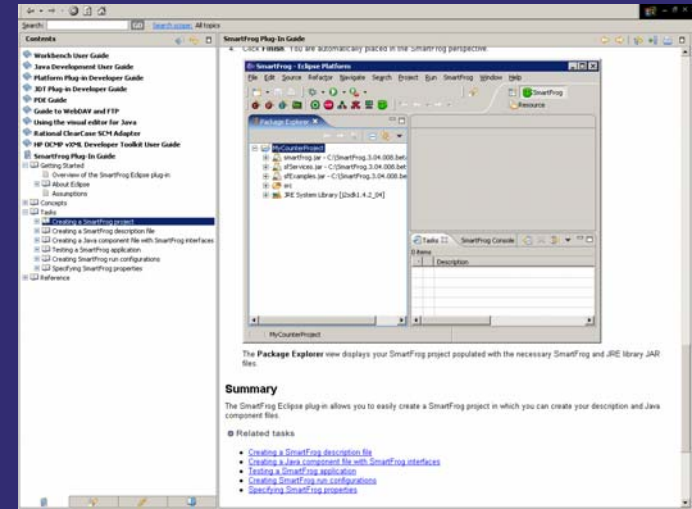
- Create online help content
- Implement context sensitive help
- Reference existing documentation files

## How do you do it?:

- Follow the Eclipse model to create HTML content files and XML configuration files to produce an online guide whose content is accessible via the table of contents defined by your XML files.
- Use the setHelp method in the WorkbenchHelp static class to associate a context id with a Control, IAction, Menu, or MenuItem.
- Provide links to any existing documents by providing their URLs in the appropriate XML configuration file.

## Why we do that in SmartFrog?:

- Nice customer collateral





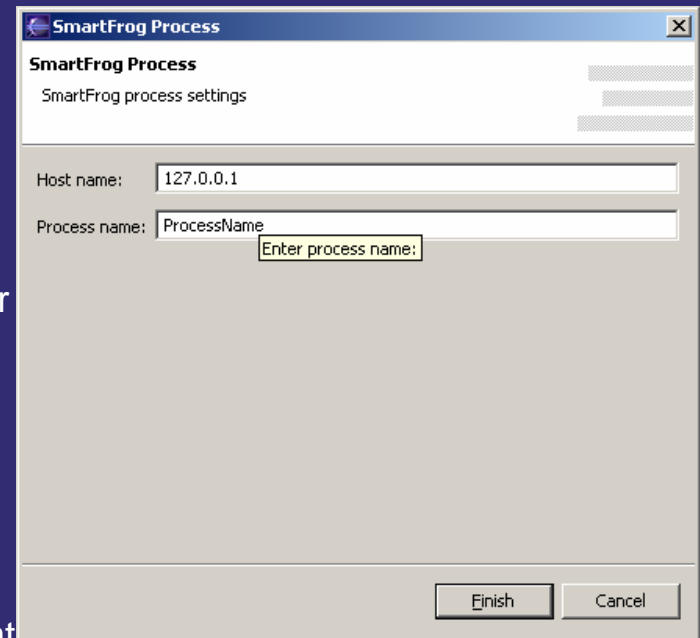
# Deploy

- **What can you do?:**
  - Deploy application to the remote/local server
  - Setup the configuration
  
- **How do you do it?:**
  - Use Eclipse WebDAV/Ftp plugin to support copying files to remote site
  - Create your own exporting wizard (configuration and deploy)
  - Add a button to call external tool (legacy utility)

## Why we do that in SmartFrog?:

- Enough
- Can easily to extend with Exporter later

JG4



JG4

In the case of SF how to you lik with its particular deployment infrastructure (sfdaemon).

Julio G, 1/26/2005

# Demo

- [www.smartfrog.org](http://www.smartfrog.org)

# Demo!

# Summary

To rapidly develop your language in Eclipse (What else?)

1. Use the above steps, to select the solution that fits your needs.
2. Make sure to cover the necessary development steps.
3. Implement the basic features that other features depend on first.
4. Leverage your existing components, use Eclipse as an integration tool to integrate those components, and be prepared to replace those components with Eclipse native components later.
5. Stick with Eclipse suggested API to make your plug-in is easy to upgrade;
6. Use XML as the language format. (very easy to add content outline, code completion .., with exiting plug-in).

# Q and A