JavaOne℠

Sun's 2002 Worldwide Java Developer Conference

A Framework for Assembling Apps From Modular Components

Making Life Easier

Dave Wilson
Architect
Portal Software, Inc.

Overall Presentation Goal

Learn how to make life easier for your developers with modular development, and learn how to make life easier for your customers by combining multiple separate application modules into “Centers”

Learning Objectives

- As a result of this presentation, you will be able to:
  - Make user interface decisions about combining components into a single application
  - Design a Java™ technology-based framework ("Java framework") that allows components to be built separately, and then combined
  - Use properties files to define how components should be combined

Speaker's Qualifications

- Dave Wilson
  - Architect at Portal Software (Cupertino, CA and Hamburg, Germany)
  - Have used many applications and problem—domain frameworks for over the last 14 years
  - Have written a number of my own frameworks using Object Pascal, C++, and the Java programming language
  - I used the BMF framework described today to help develop an integrated suite of developer tools

Co-authors' Qualifications

- Raghu Venkateshwaran
  - Staff Engineer at Portal Software (Cupertino)
  - Developer of the BMF framework
- Guido Krüger
  - Principal Engineer at Portal Software (Hamburg)
  - Java framework and library developer
  - Author of best-selling books for Java™ technology in Germany

Are Your Customers Confused, Frustrated, and Angry?

JavaOne

Are Your Customers Confused, Frustrated, and Angry?

- Your job is not to just write software

Are Your Customers Confused, Frustrated, and Angry?

- Your job is not to just write software
- Your job is to make life simple for your customer

Are Your Customers Confused, Frustrated, and Angry?

- Your job is not to just write software
- Your job is to make life simple for your customer
- Large collections of unrelated tools make life more difficult for customers. This makes customers frustrated (and angry)

Are Your Customers Confused, Frustrated, and Angry?

- Your job is not to just write software
- Your job is to make life simple for your customer
- Large collections of unrelated tools make life more difficult for customers. This makes customers frustrated (and angry)
- Integrating your tools into functional "centers" of activity can make your customers happier and more productive

Are Your Customers Confused, Frustrated, and Angry?

- Your job is not to just write software
- Your job is to make life simple for your customer
- Large collections of unrelated tools make life more difficult for customers. This makes customers frustrated (and angry)
- Integrating your tools into functional "centers" of activity can make your customers happier and more productive

Presentation Agenda

- Market requirements (motivation)
- Terminology (what is a component???!!?)
- Phase I: **B**usiness **M**anagement **F**ramework
  - Live demo
  - GUI design
  - Framework architecture
  - Framework implementation
  - Lessons learned
- Phase II: **P**ortal "**J**ava **F**ramework"
  - Work in progress
  - Demo of prototype
  - Architecture

Market Requirements

- Portal's Infranet Customer Management and Billing system requires tools for:
  - Installation and Configuration
  - Schema and Code customization
  - Defining price plans
  - Customer management
  - Bulk loading of batch usage data
  - Billing
  - Accounts Receivable
  - Etc.
- Complex system; long-learning curve

Market Requirements

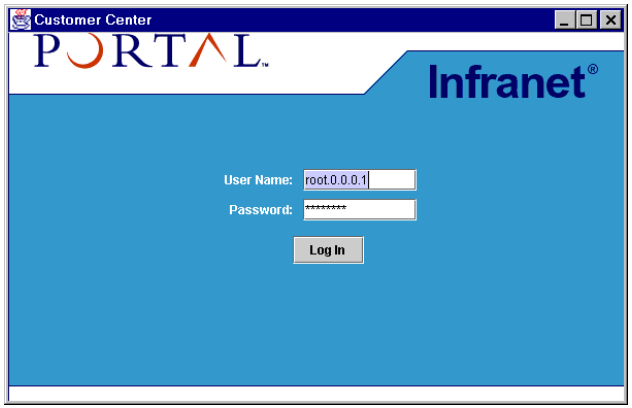
- Support multiple client platforms
  - Solaris, HP-UX, IBM AIX, Windows
  - Solution: Java technology
- Provide easy-to-use GUIs
  - Solution: Java Foundation Classes (JFC/SWING) API
- Support for world-wide customer base: I18N, L10N
  - Solution: Java technology resources and careful attention to detail

Market Requirements

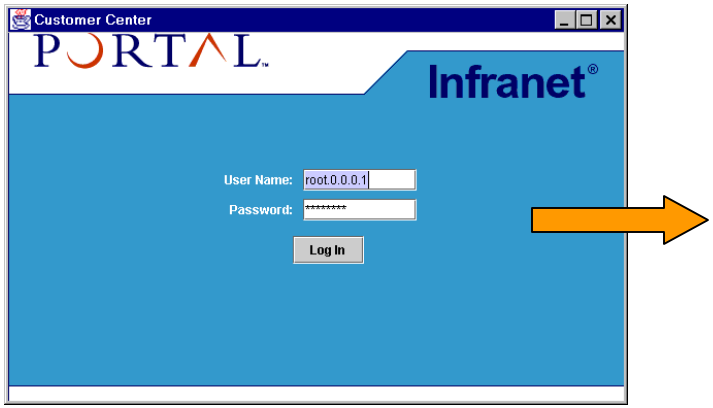
- Manage complexity for our developers and customers
  - Solution: Custom framework for combining related components into application “Centers”
  - Customer Center
  - Developer Center
  - Pricing Center
  - Etc.

Bad: Too Many Separate Applications w/Separate Logins

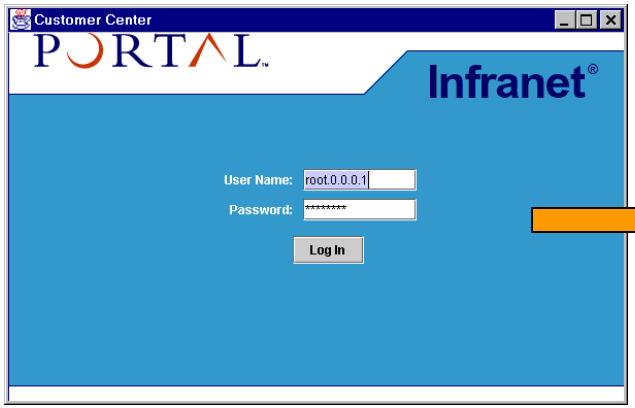
Bad: Too Many Separate Applications w/Separate Logins

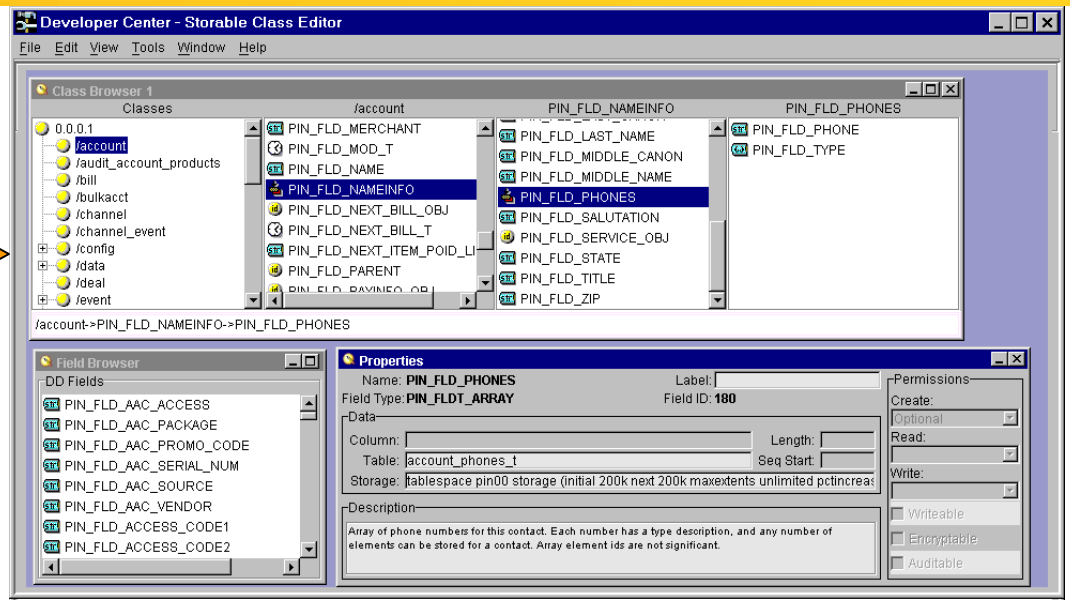


Bad: Too Many Separate Applications w/Separate Logins

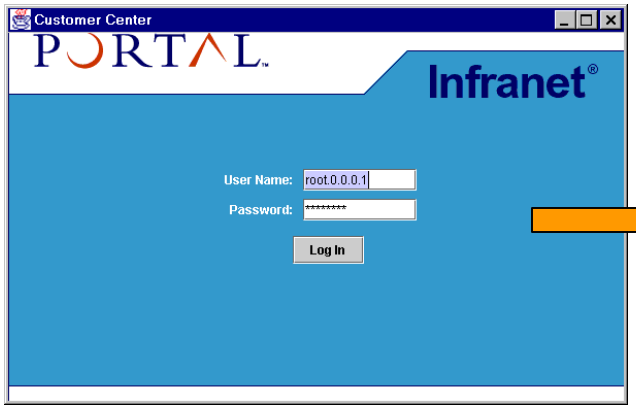


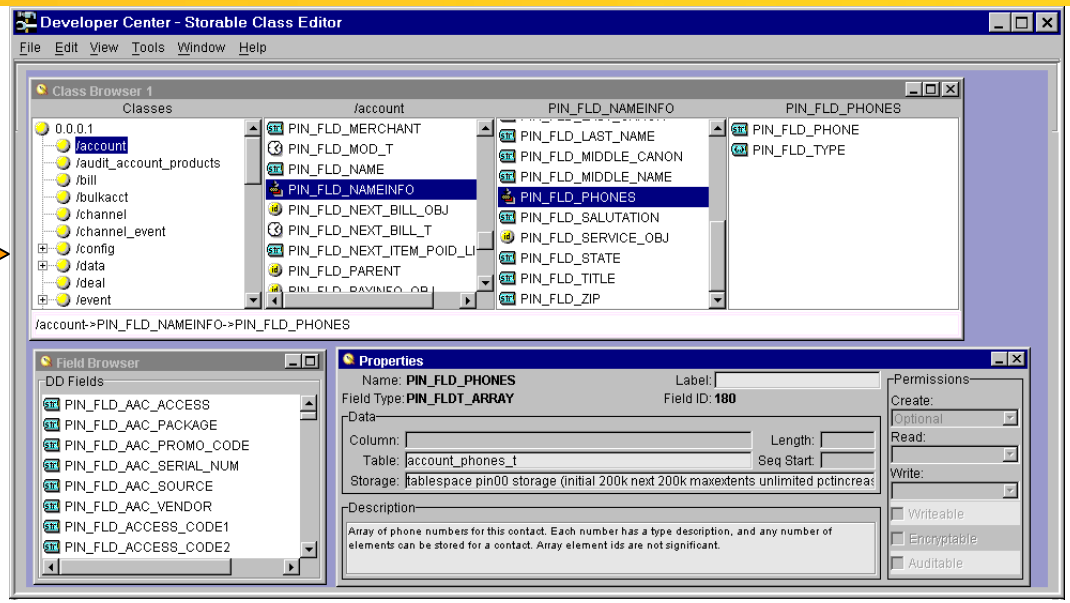
Bad: Too Many Separate Applications w/Separate Logins

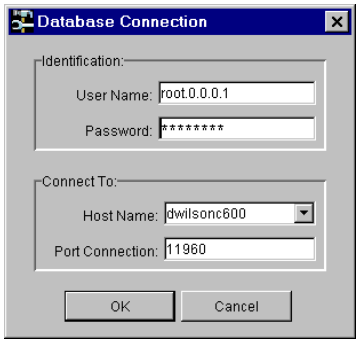




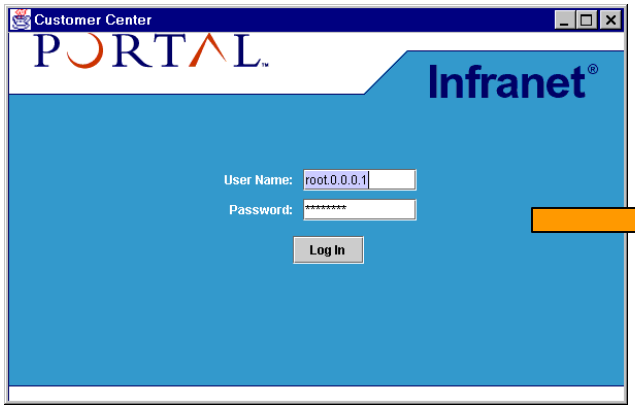
Bad: Too Many Separate Applications w/Separate Logins

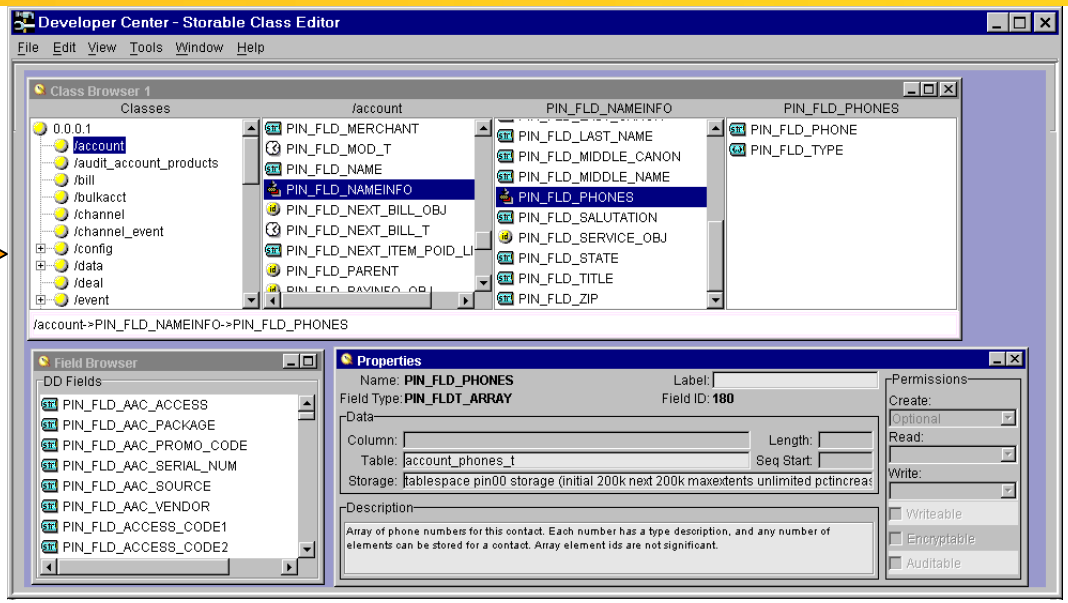


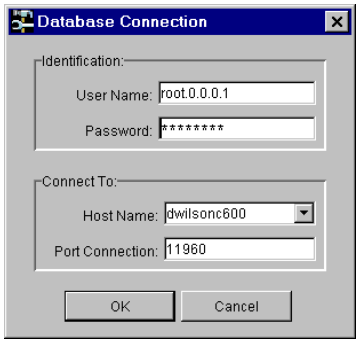




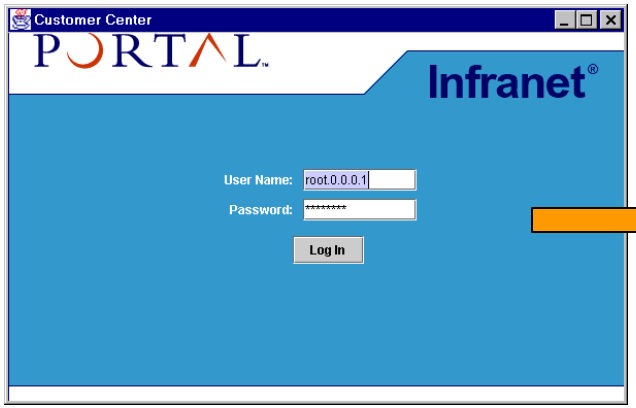
Bad: Too Many Separate Applications w/Separate Logins

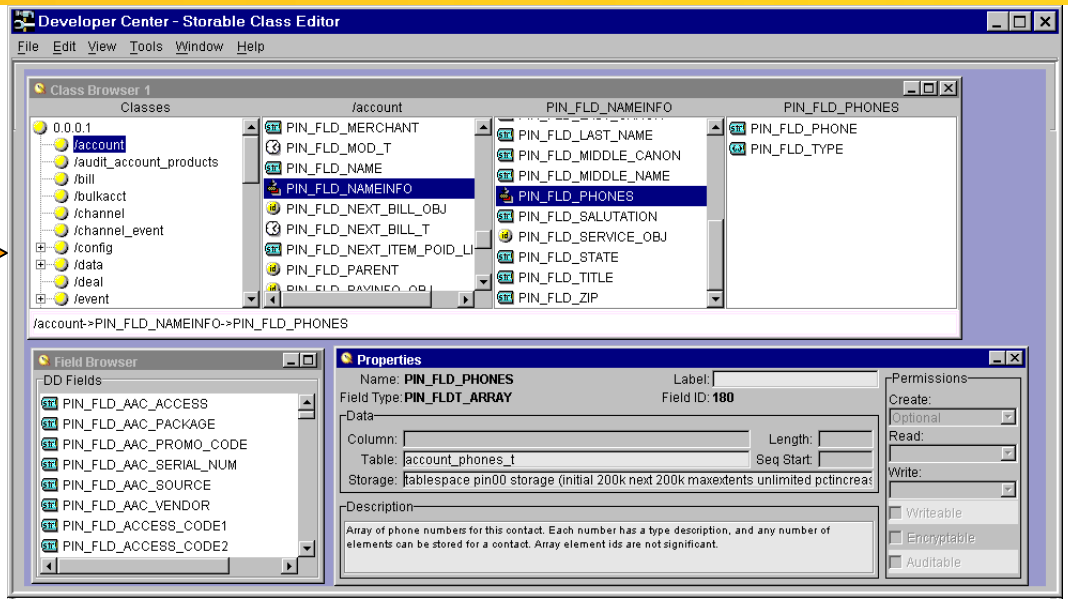


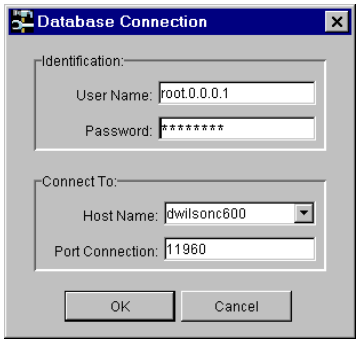


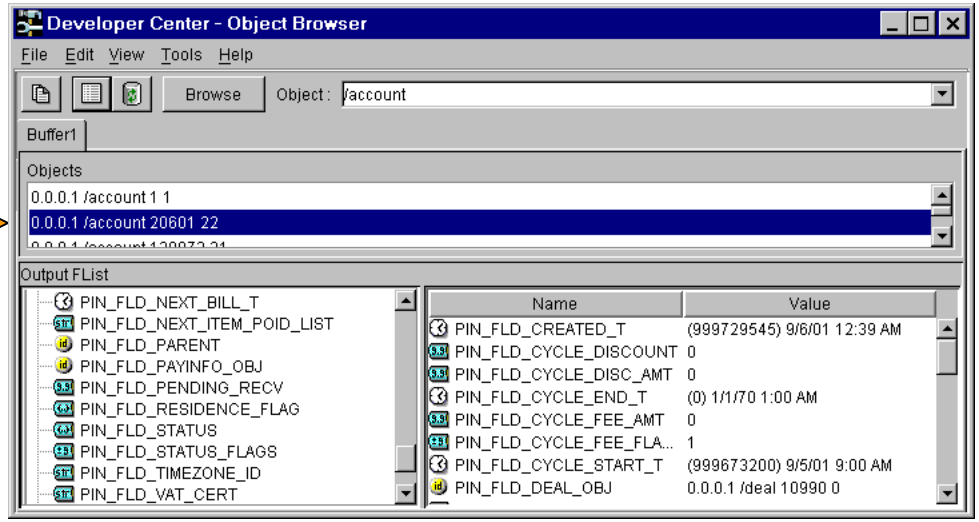


Bad: Too Many Separate Applications w/Separate Logins

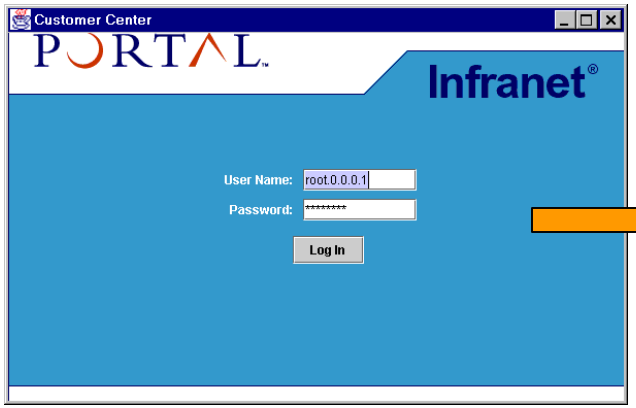


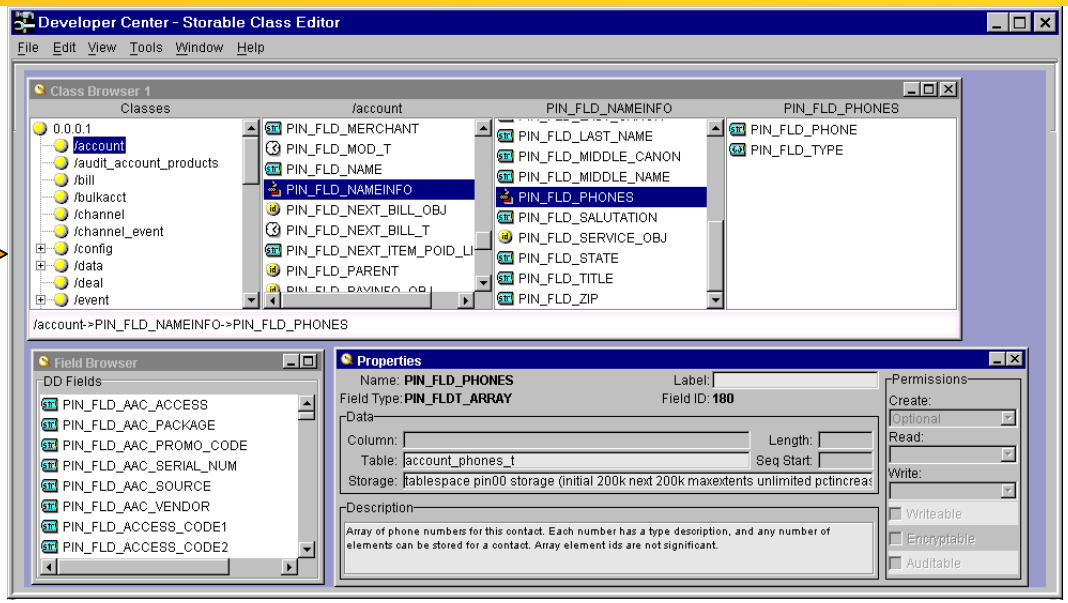


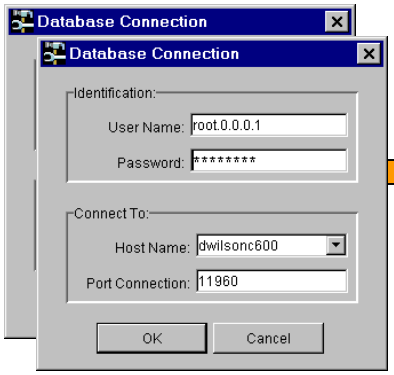


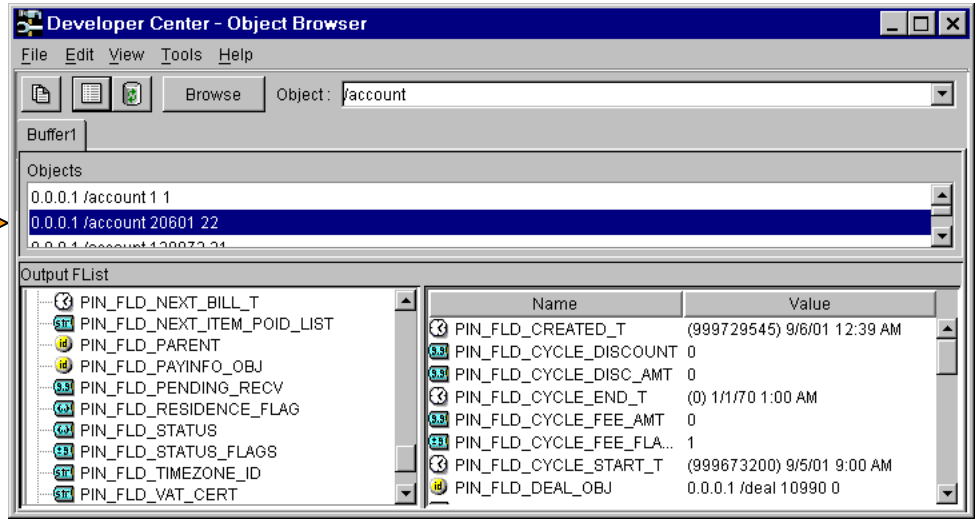


Bad: Too Many Separate Applications w/Separate Logins

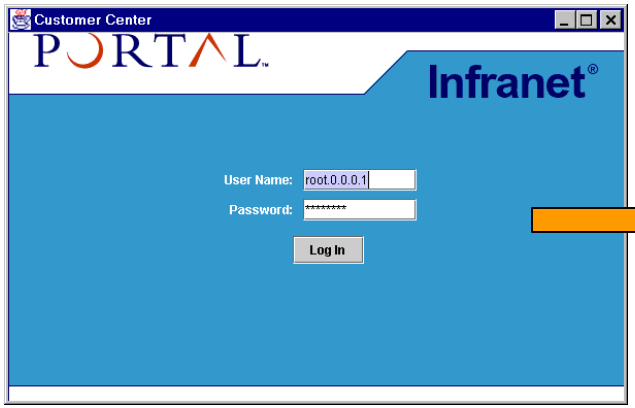


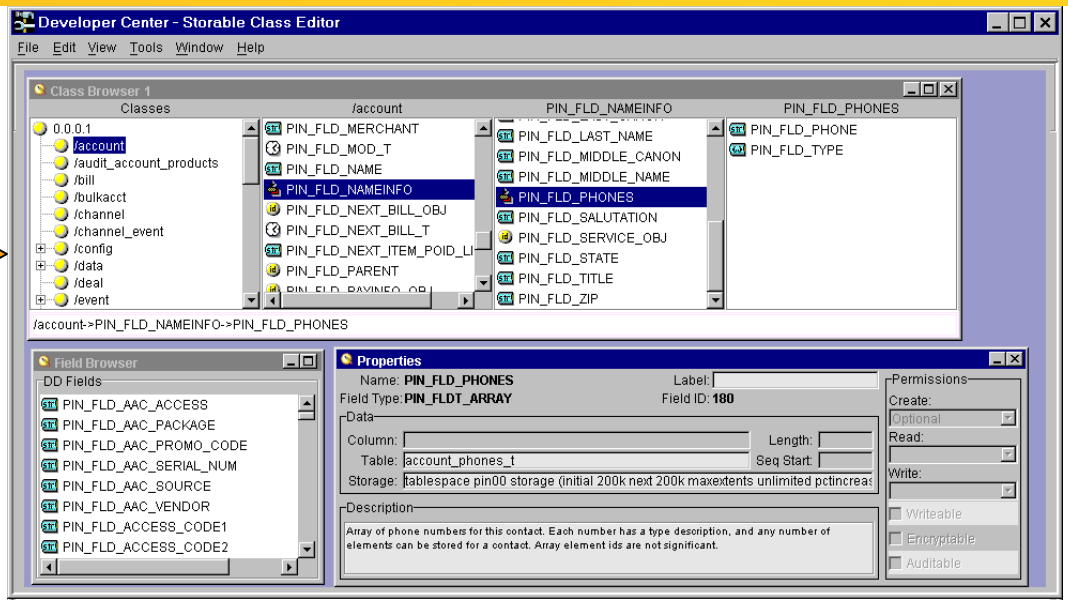


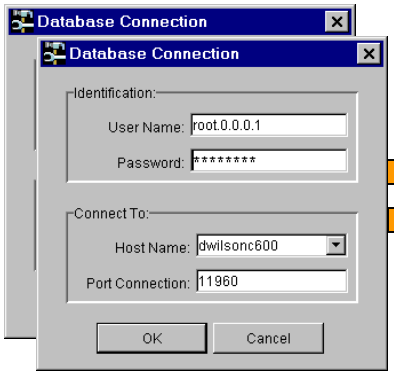


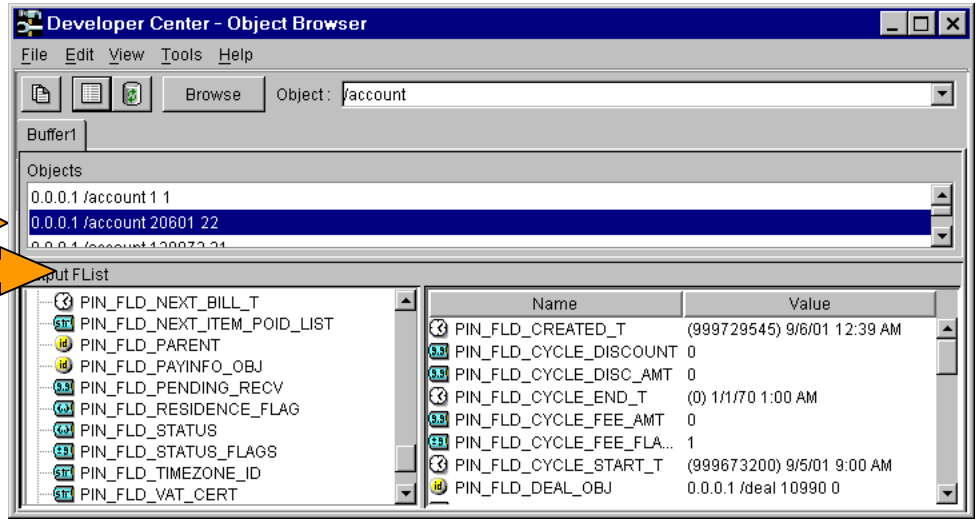


Bad: Too Many Separate Applications w/Separate Logins

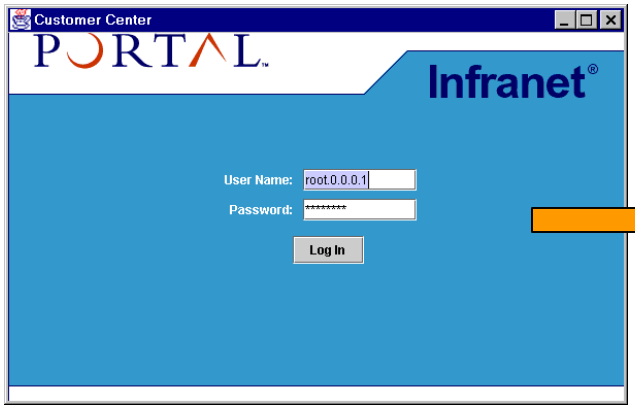


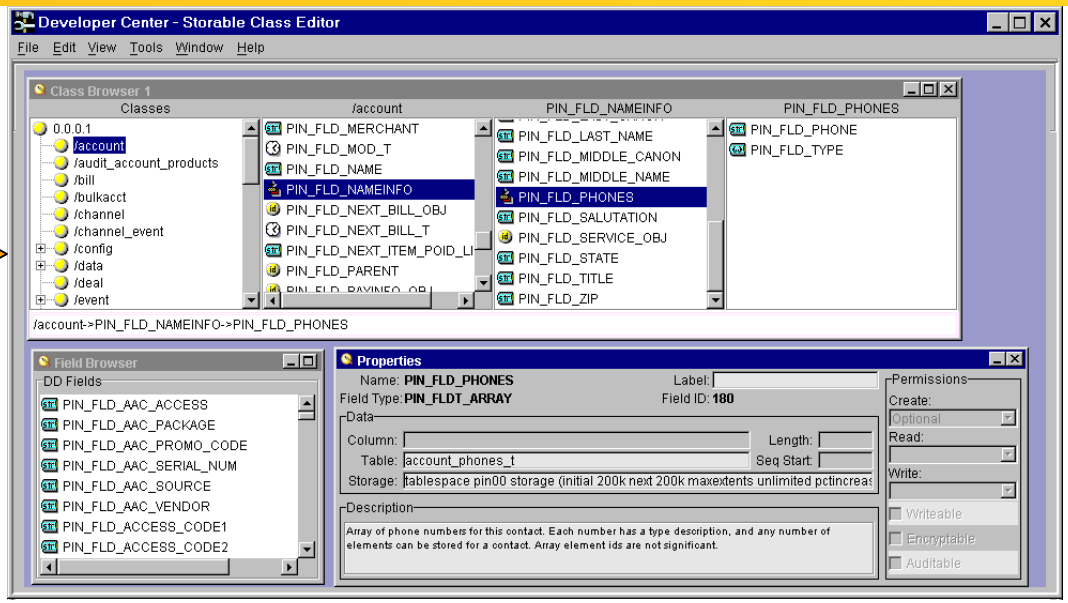


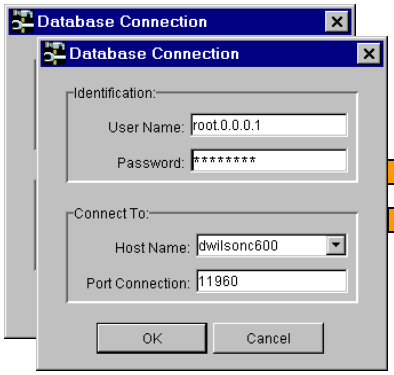


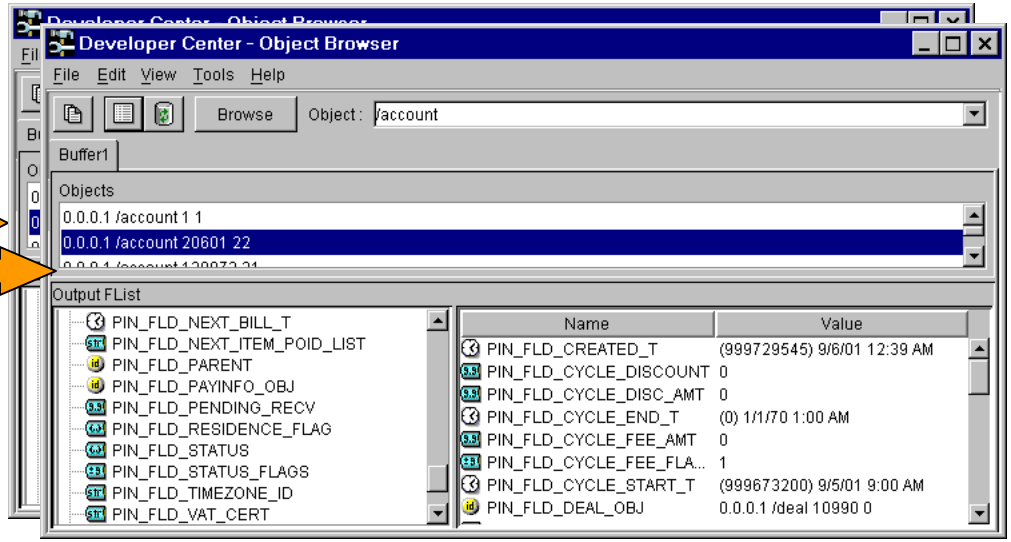


Bad: Too Many Separate Applications w/Separate Logins



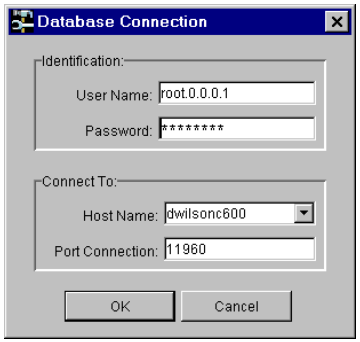




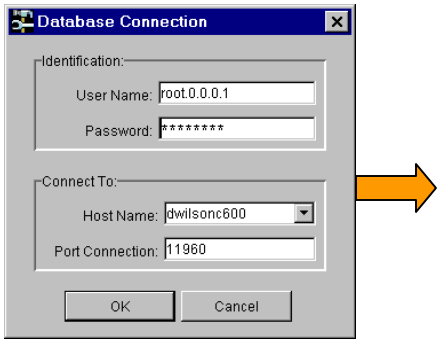


Good: “Developer Center”: One Application = Four Modules

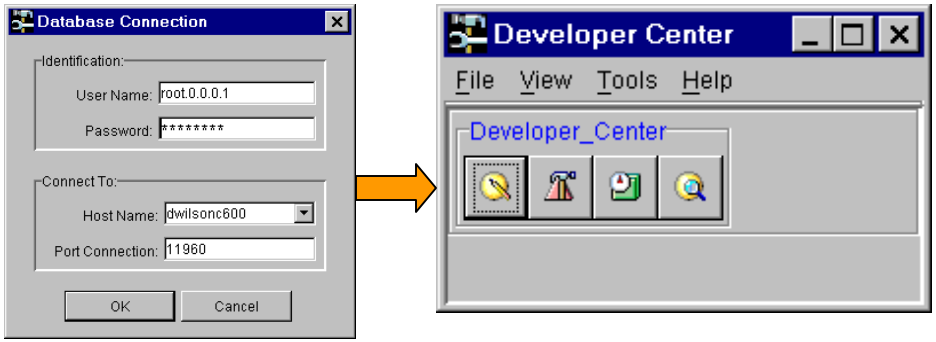
Good: “Developer Center”: One Application = Four Modules



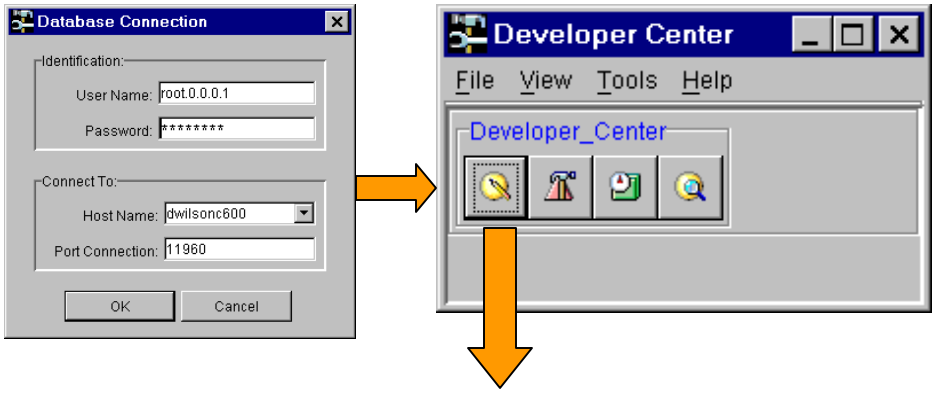
Good: “Developer Center”: One Application = Four Modules



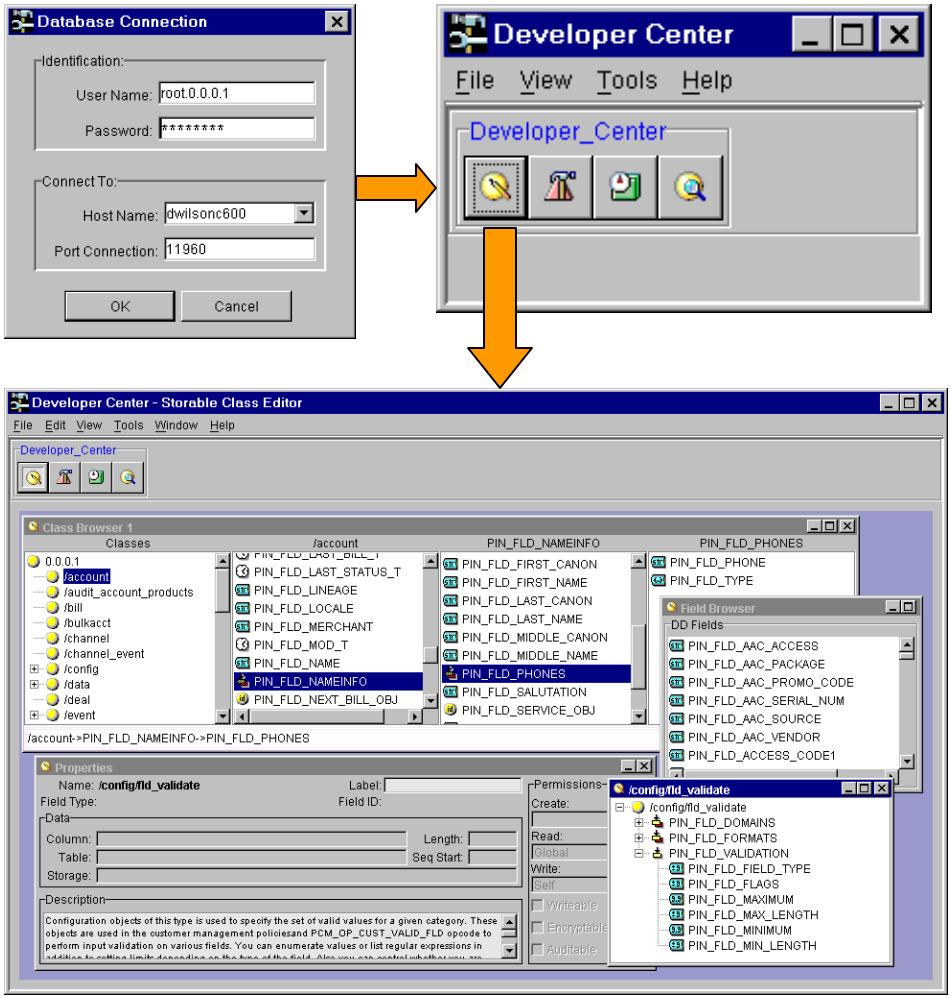
Good: “Developer Center”: One Application = Four Modules



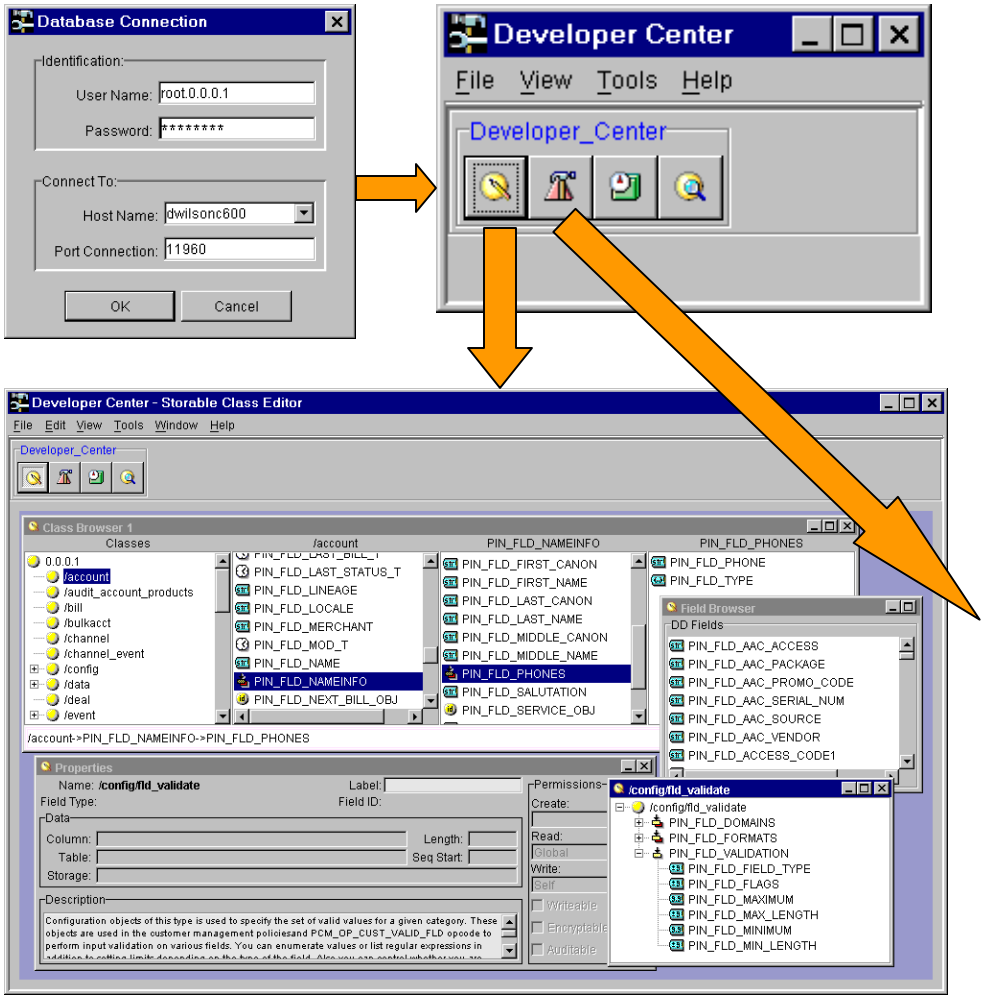
Good: “Developer Center”: One Application = Four Modules



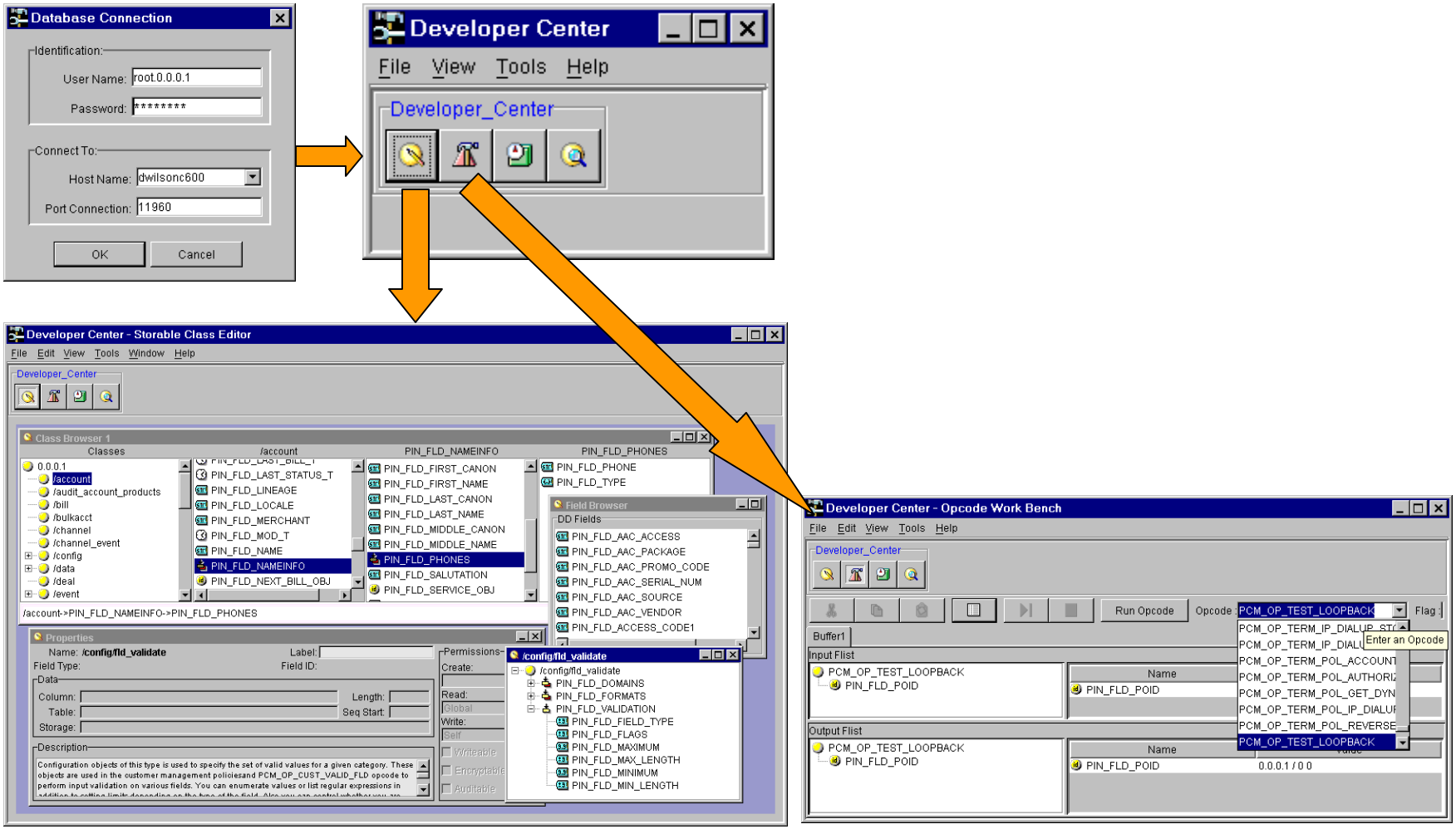
Good: “Developer Center”: One Application = Four Modules



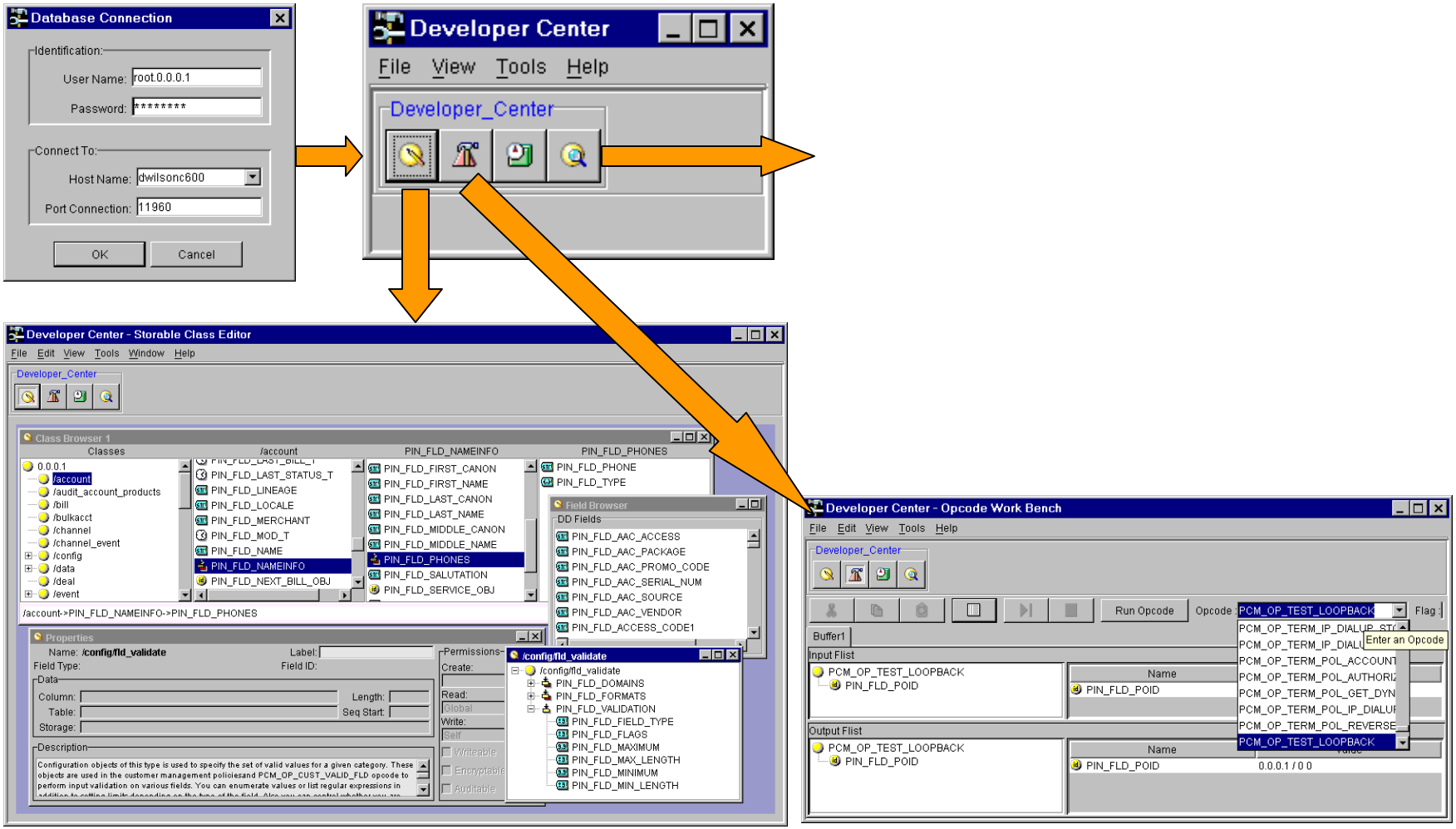
Good: “Developer Center”: One Application = Four Modules



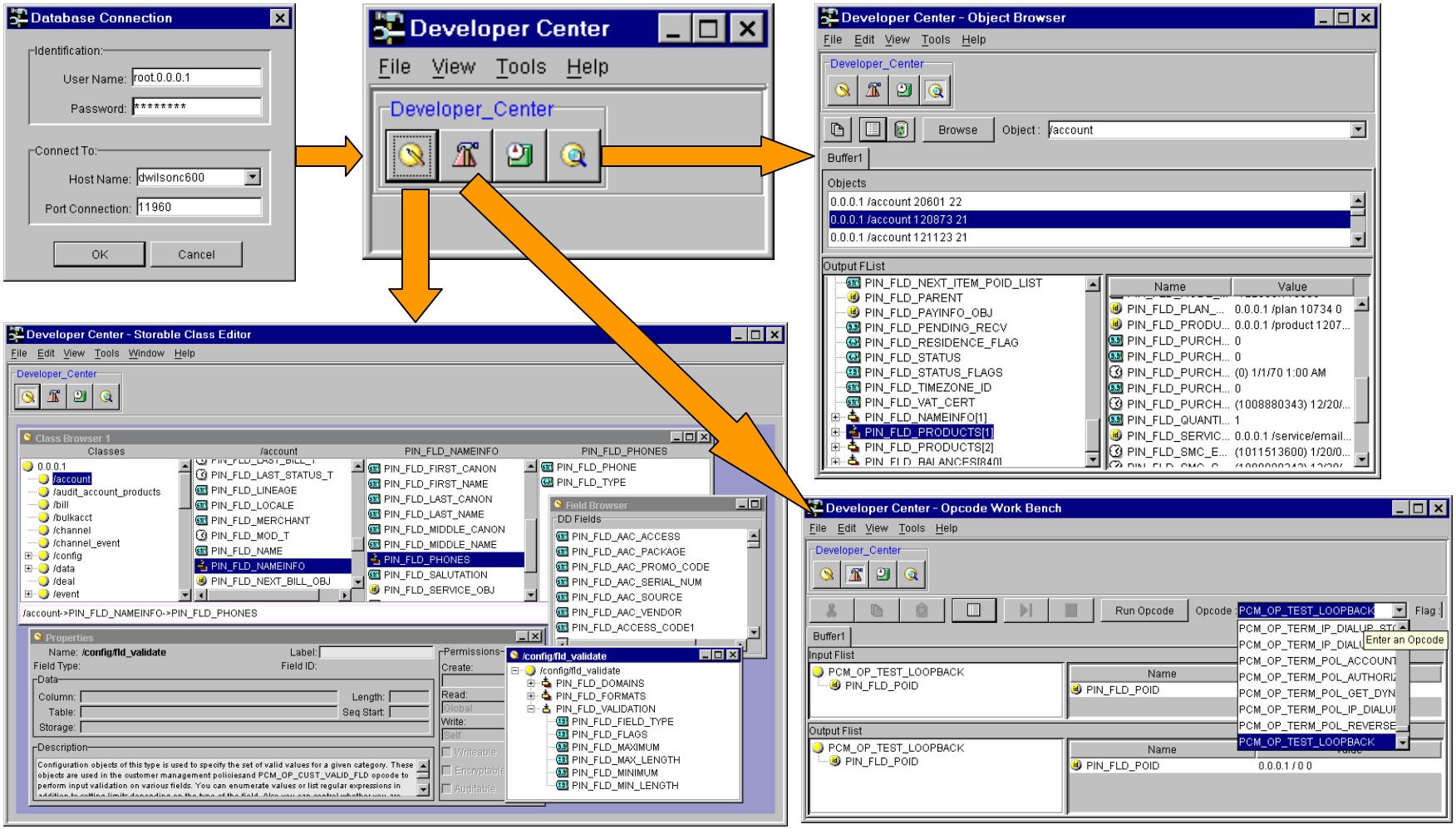
Good: “Developer Center”: One Application = Four Modules



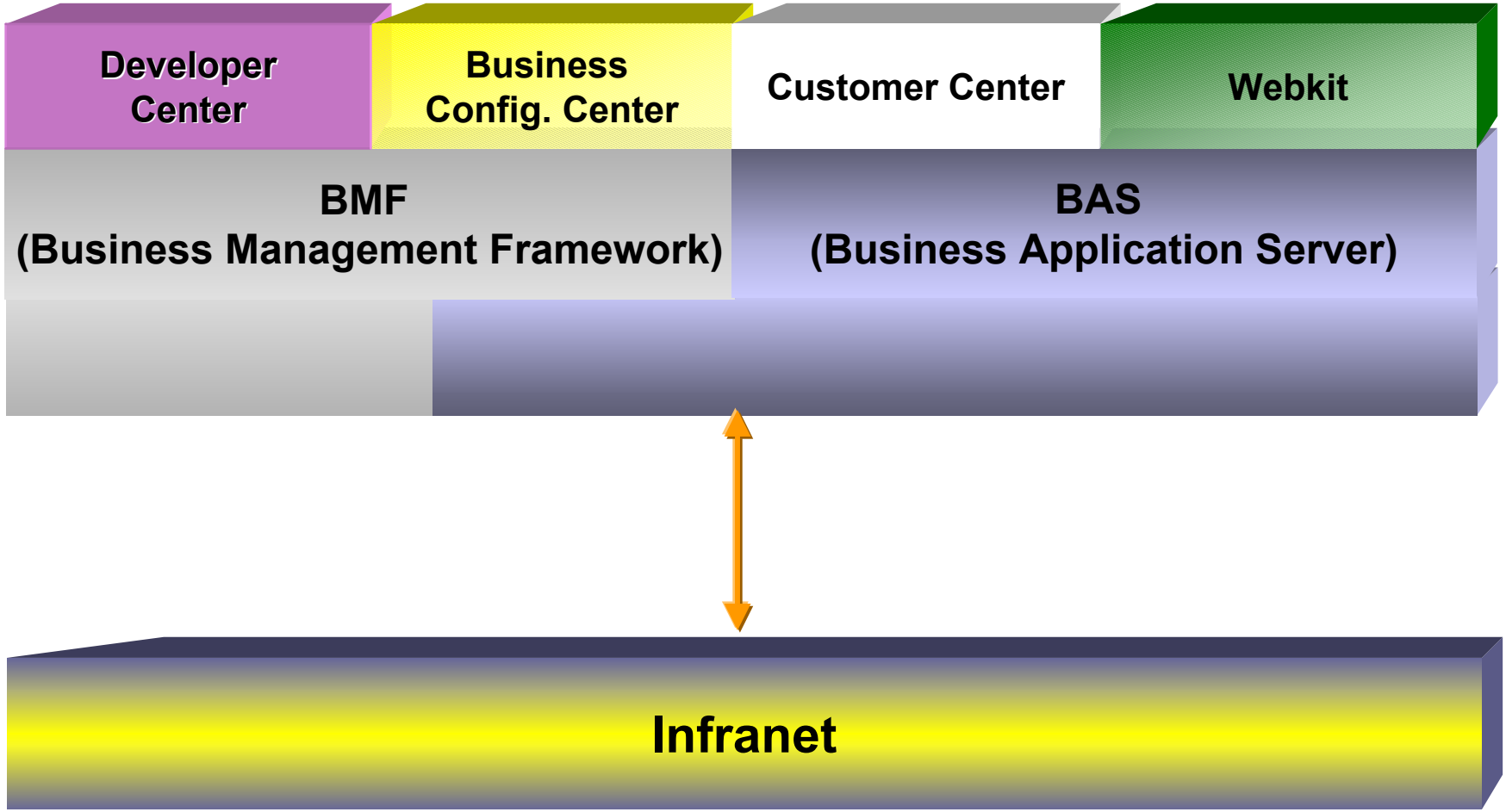
Good: “Developer Center”: One Application = Four Modules



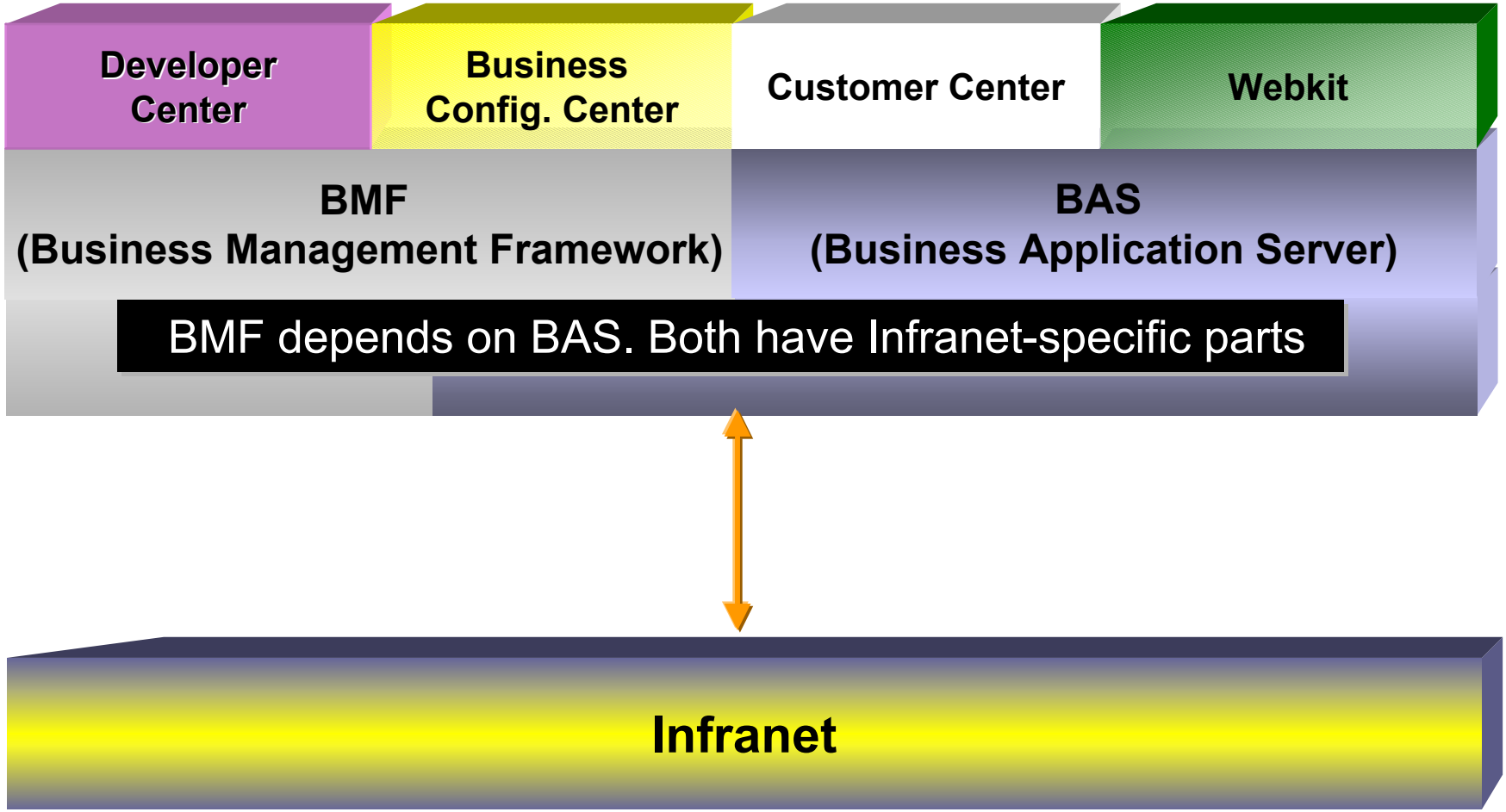
Good: “Developer Center”: One Application = Four Modules



Infranet Client/Server Architecture BMF and BAS Client Frameworks



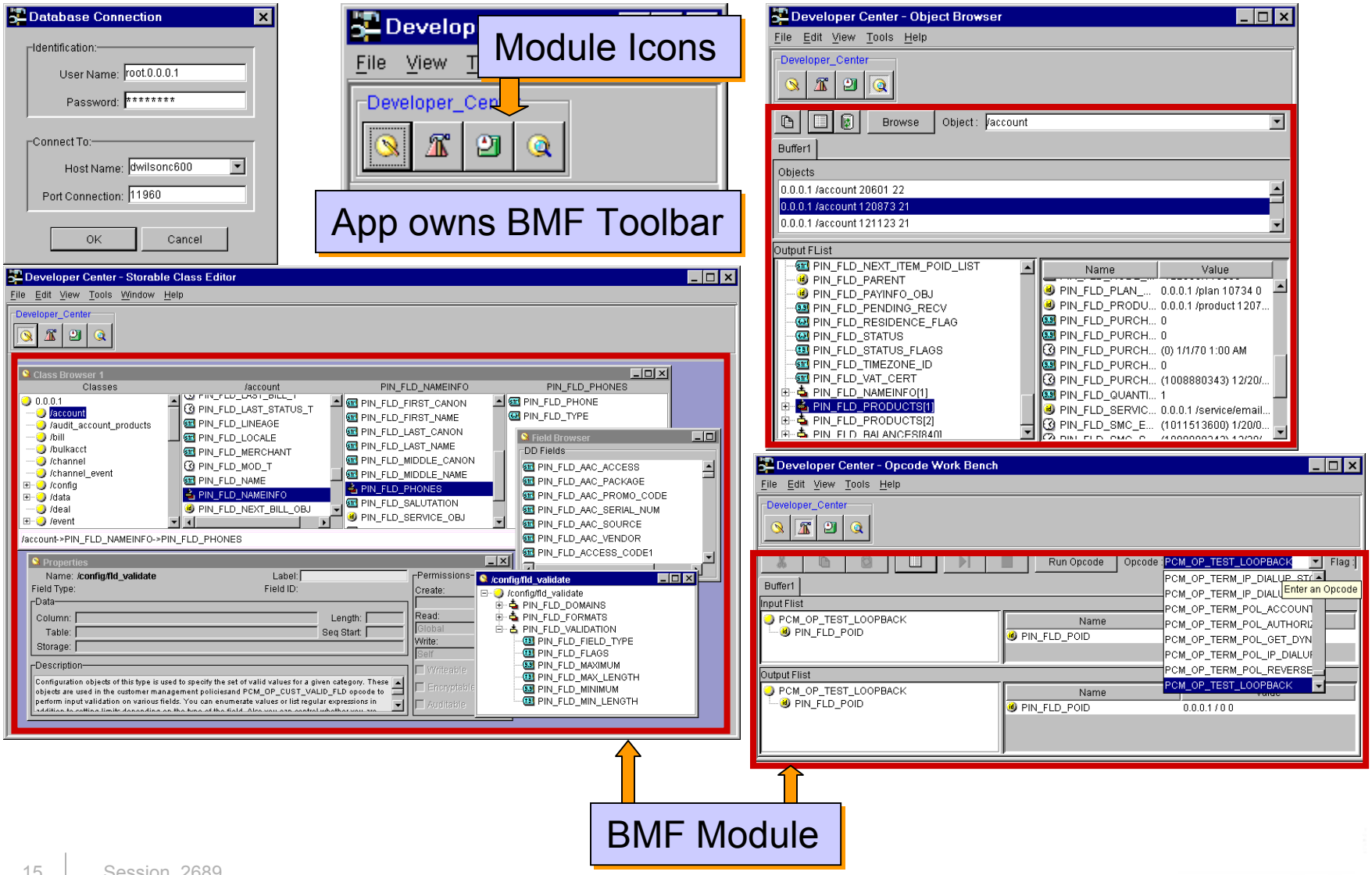
Infranet Client/Server Architecture BMF and BAS Client Frameworks



I. BMF Features and Benefits

- Provides a pluggable module architecture
- Provides an Infranet login module and obtains the Infranet connection
- Creates and manages Menu bar, toolbar and status bar
- Takes care of application Help, About, etc.
- Supports some product-specific features

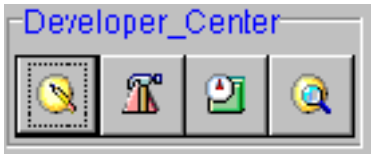
Terminology



Demo

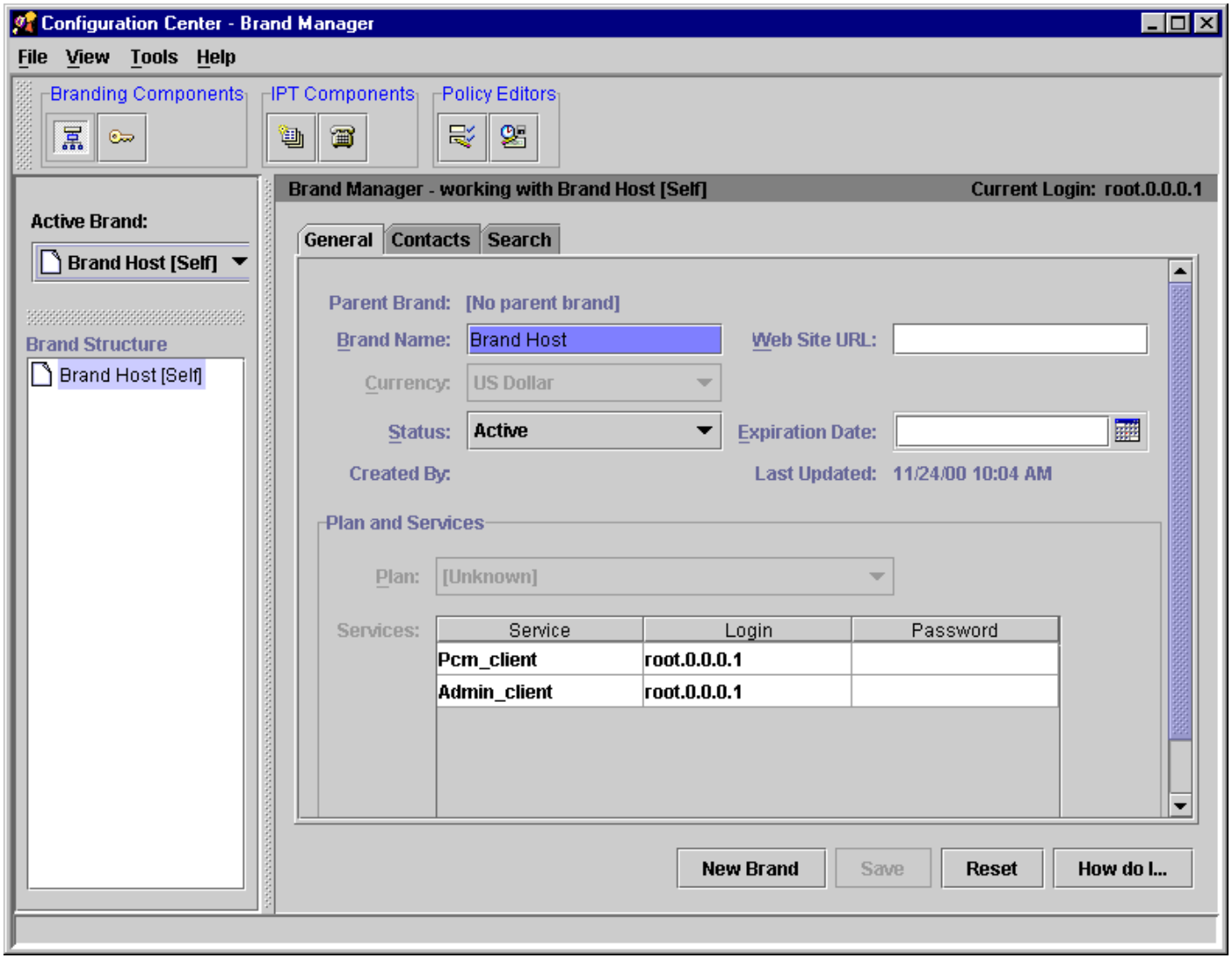
I.Developer Center

I. Visual Design Choices



- Toolbar contains an icon for each module
- Related icons can be grouped
- Only one module body is visible at a time
  - Select visible module by clicking on toolbar icon, or selecting menu item
- Menu bar switches to show menus for visible module
- Status area on bottom
- Module contains a JComponent or JPanel
  - Can do anything you want inside the component
  - Other panels, tabbed panes, internal frames, tables, etc.

I. Example BMF Application



Designing an Applications Framework

- How does someone run the application?
  - Is there a “manager” object?
- What classes should the developer instantiate?
  - Public methods to call?
- What is the default behavior? Where is the flow of control?
- How is the framework customized?
  - Data-driven (properties files, XML files, etc.)
  - Subclassing and overriding
- What should the developer subclass
  - Protected methods to override?
- What developer methods are called when the user does things?

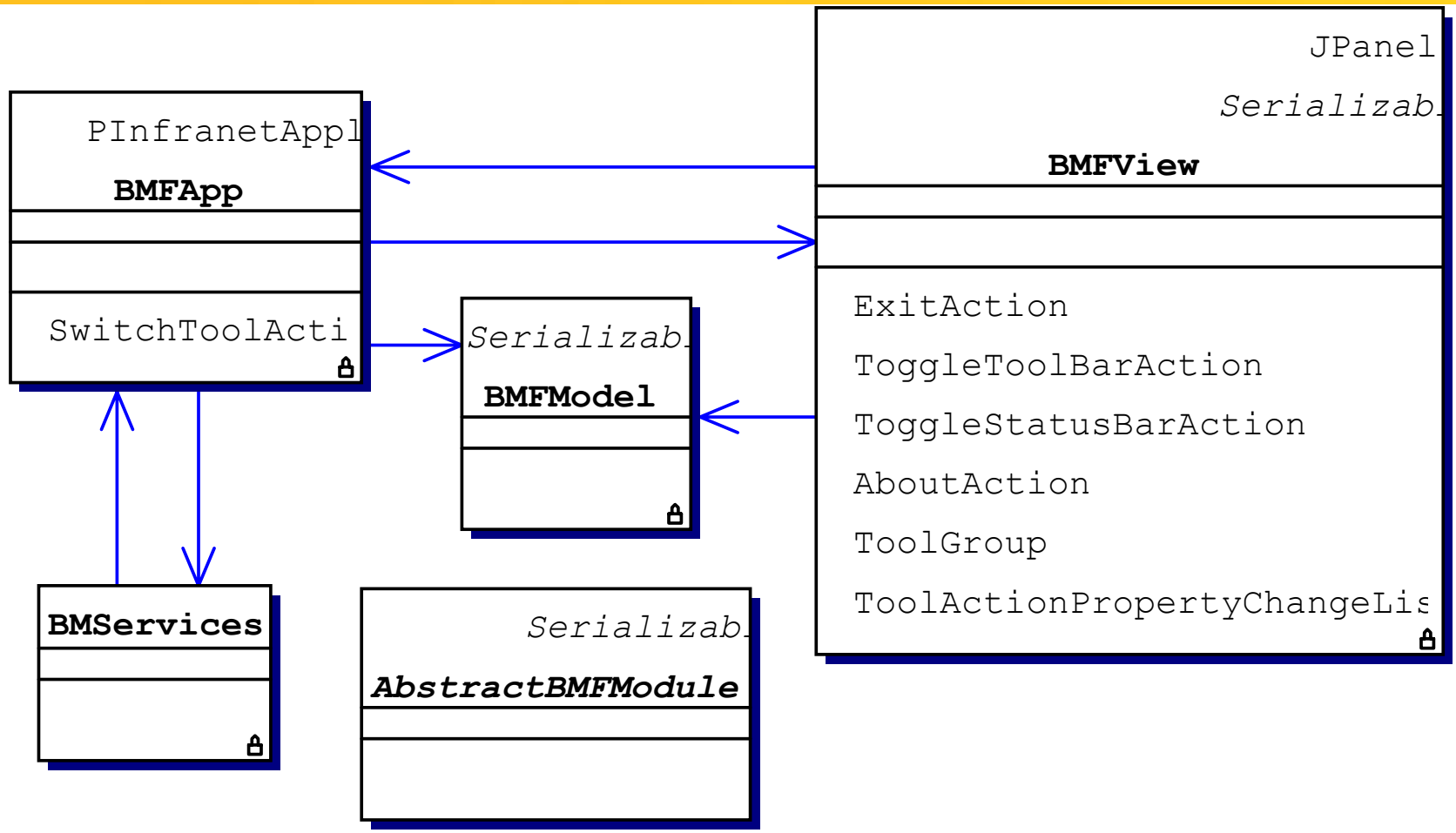
I. Architectural Decision: Application

- The framework provides one concrete application class, **`BMFApp`**
  - It is the heart of the framework
  - It manages all the modules (with help from a few other classes such as **`BMFModel`**)
  - It is not subclassed
  - It calls methods of installed modules as necessary

I. Architectural Decision: Modules

- Each module developer must provide a concrete subclass of **`AbstractBMFModule`**
  - The module must return a “”root view” which is a **`JComponent`**. Do anything in the root view
  - There are **`protected`** methods called by the framework when a module starts, stops, etc. Override if necessary
  - The module must return a **`Vector`** of actions, used by menu items
- Modules can call upon an instance of **`BMServices`** for utility functions

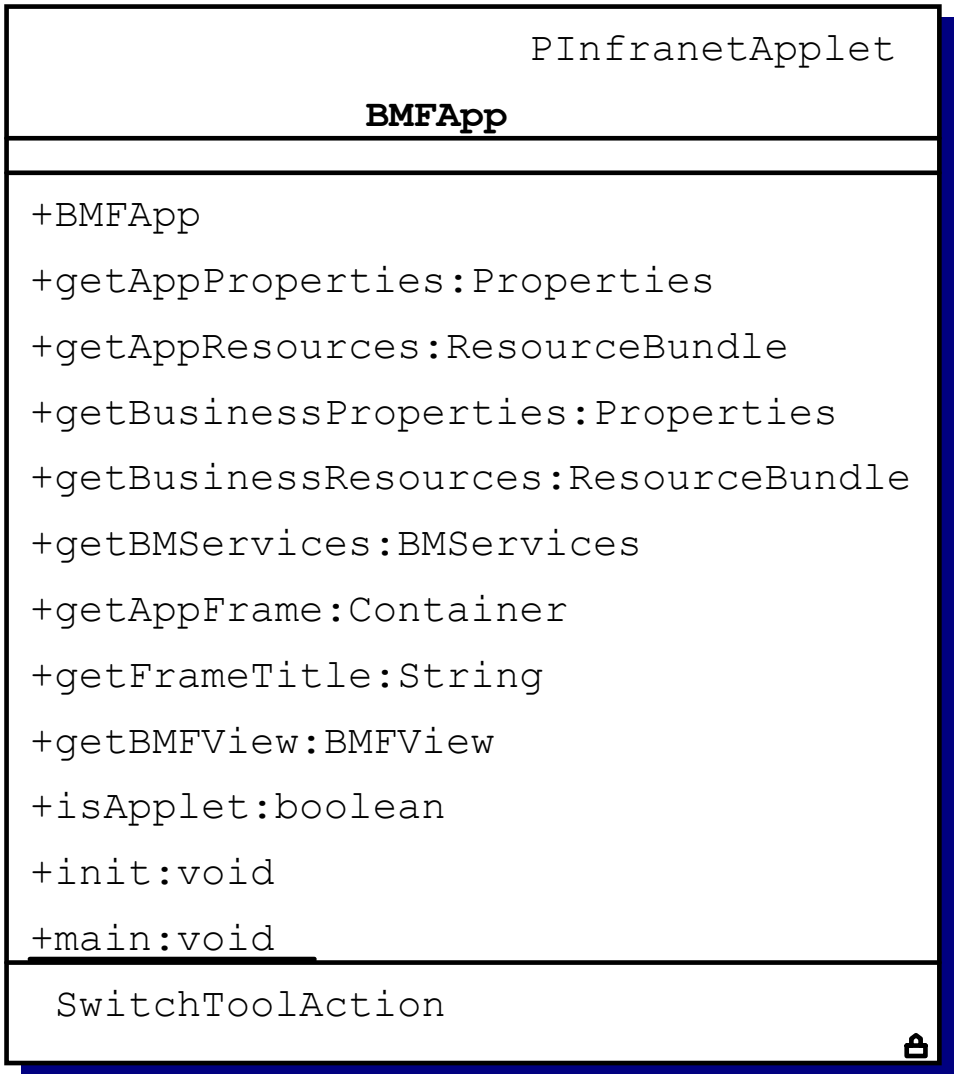
I. Framework Architecture Based on Model-View-Controller



"Module" = "Tool"

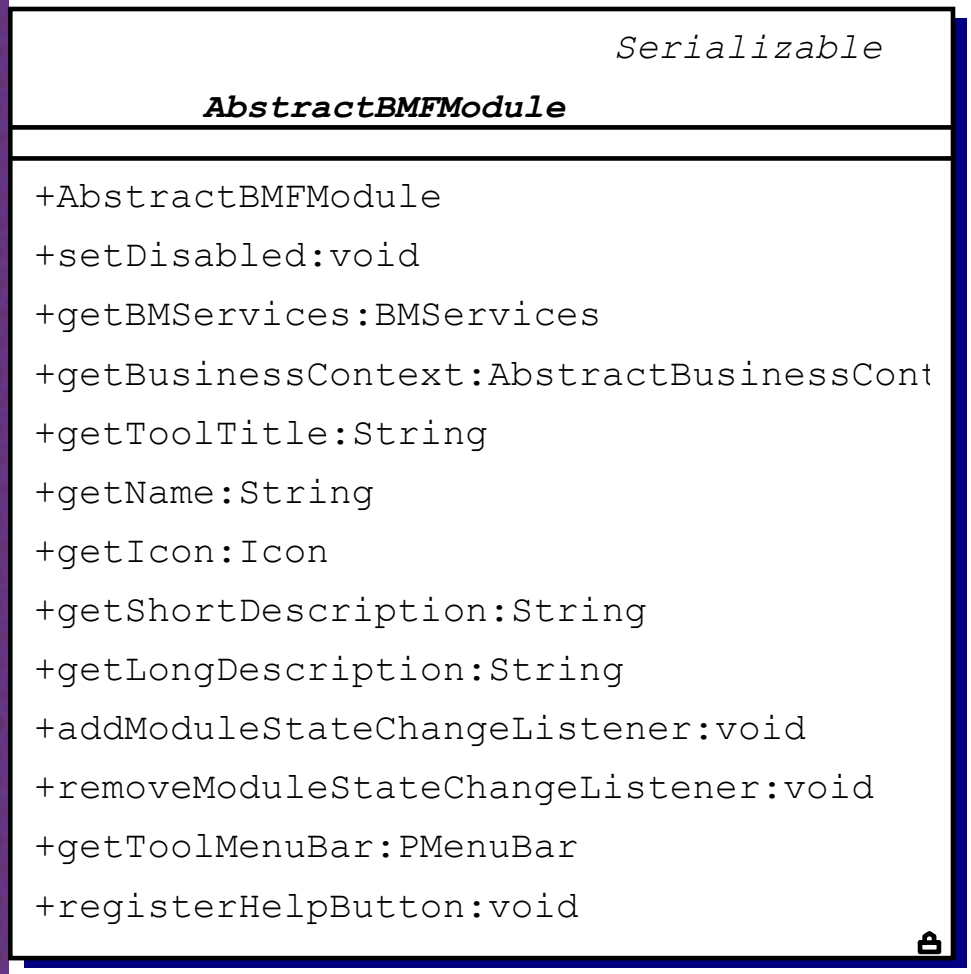
JavaOne

Concrete Application Class Public Methods



- The user runs an instance of **BMFApp**
- It controls all the modules
- It has a few public methods that can be called by module developers
- It has protected methods to run the framework

Abstract Module Class Public Methods



- Each module developer must implement this module
- There is one protected abstract method to implement
- There are many optional empty protected methods to override for customization (see next slide)

I. Framework Architecture
Some Module Protected Overrides

```
// initialize the module
Vector getMenuActions()                          // return menu actions
abstract JComponent getToolRootView()            // return main view
boolean init()                                   // called after login
void start()                                     // start threads here

// called during normal operation
void activate()                                  // called when activated
void deactivate()                                // called when deactivated
void fireModuleStateChangeEvent(Object eventData)

// called as user wants to quit
boolean canExit()                                // OK to exit app?
boolean isDirty()                                // Save on switch?
boolean commit()                                 // Update user data
void stop()                                      // Stop threads, etc. here
void exit()                                      // called as app quits

// provide module status
boolean isActivated()
boolean isHidden()
boolean isDisabled()
```

I. Framework Architecture
Some Module Protected Overrides

```
// initialize the module
Vector getMenuActions()                          // return menu actions
abstract JComponent getToolRootView()            // return main view
boolean init()                                   // called after login
void start()                                     // start threads here

// called during normal operation
void activate()                                  // called when activated
void deactivate()                                // called when deactivated
void fireModuleStateChangeEvent(Object eventData)

// called as user wants to quit
boolean canExit()                                // OK to exit app?
boolean isDirty()                                // Save on switch?
boolean commit()                                 // Update user data
void stop()                                      // Stop threads, etc. here
void exit()                                      // called as app quits

// provide module status
boolean isActivated()
boolean isHidden()
boolean isDisabled()
```

- Only one abstract method
- Developer overrides others as needed

Demo

Framework in Action

I. Architectural Decisions: Resources

- The application must provide two resource files that contain all user-visible strings, and various application parameters
  - Strings for About dialog
  - Menu names
  - Menu item names, and mapping to Actions
  - Runtime decisions about layout and visibility
- Each developer create part of a pair of these resource files—a design weakness

I. Sample Module "Hello World"


```
import java.awt.Font;
import javax.swing.*;
import com.portal.bmf.*;

public class HelloWorld extends AbstractBMFModule {

    private JLabel mRootView;

    public HelloWorld () {
        mRootView = new JLabel("Hello World!!!", JLabel.CENTER);
        mRootView.setFont( new Font("SanSerif", Font.BOLD, 24));
    }

    public JComponent getToolRootView() {
        return mRootView;
    }

}
```

I. Sample Partial Resource Properties File

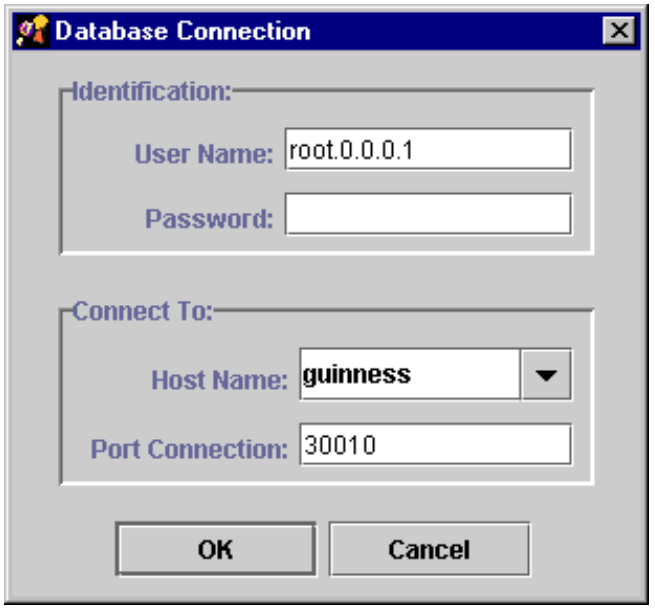
```
bmf.frame.title= Hello World
bmf.dialog.defaultTitle= Hello World
bmf.app.name= Hello World
bmf.about.image= /com/portal/bmf/images/JavaAbout.gif
bmf.about.dialogTitle=About Hello World
bmf.about.text1=Hello World, Release 6.0.2
bmf.splash.image=/com/portal/bmf/images/JavaSplash.gif
#-------------------------BMF Modules ----------------------------------
main.hello.name= Hello
main.hello.desc= Hello World
main.hello.menu= tool
main.hello.icon= /com/portal/bmf/images/Branding0.gif
main.hello.icond= /com/portal/bmf/images/Branding1.gif
#------------------------- BMF Menus ----------------------------------
bmf.menu.file = File
bmf.menu.edit = Edit
bmf.menu.view = View
bmf.menu.tool = Tools
bmf.menu.help = Help
main.open.name=    Open
main.open.desc=    Open file
main.open.menu=    file
main.open.icon=
main.open.accel=C-O
main.open.mnemonic=O
```

I. Sample Business Properties File

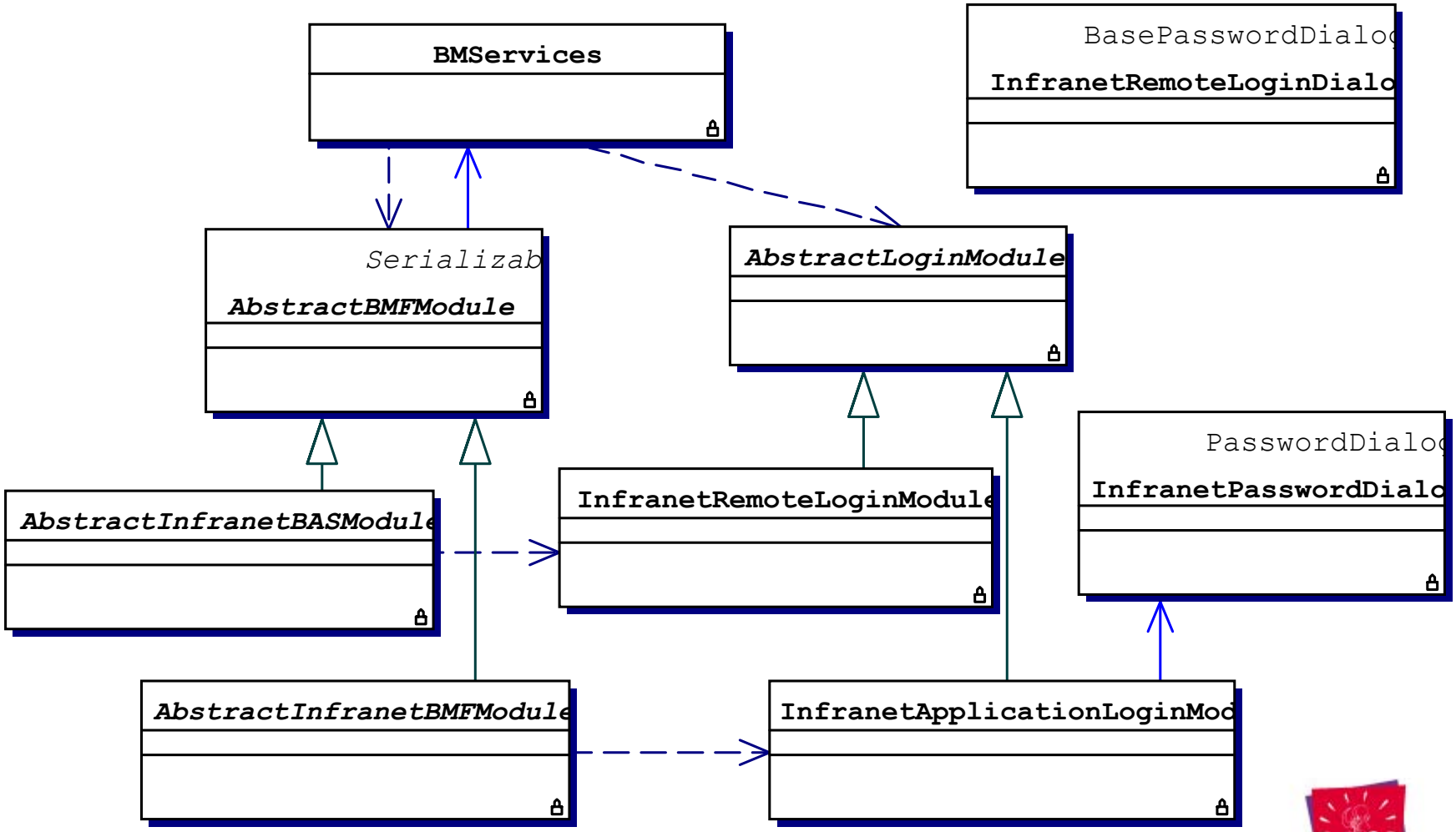
```
bmf.modules= hello
bmf.startup.module= hello
resources= hello.TSTResources
bmf.toolGroups= helloGroup
bmf.toolGroup.helloGroup= hello
module.hello.class= hello.HelloWorld
bmf.startup.looknfeel=  metal
bmf.hasLeftSidePanel= false
bmf.hasSplitPane= false
bmf.hasMenuBar= true
bmf.hasToolBar= false
bmf.toolBarIsFloatable= true
bmf.hasObjectNavigationComponent= false
bmf.objectNavigationComponentIsFloatable= false
bmf.hasStatusBar= false
bmf.WIDTH=800
bmf.HEIGHT=600
```

I. BMF Login Component

- BMF Login Component provides a connection context for the modules
- All modules can share the connection or request for separate additional connections
- You can create your own login component that logs into your server instead of Infranet



I. Framework Architecture
Login Classes

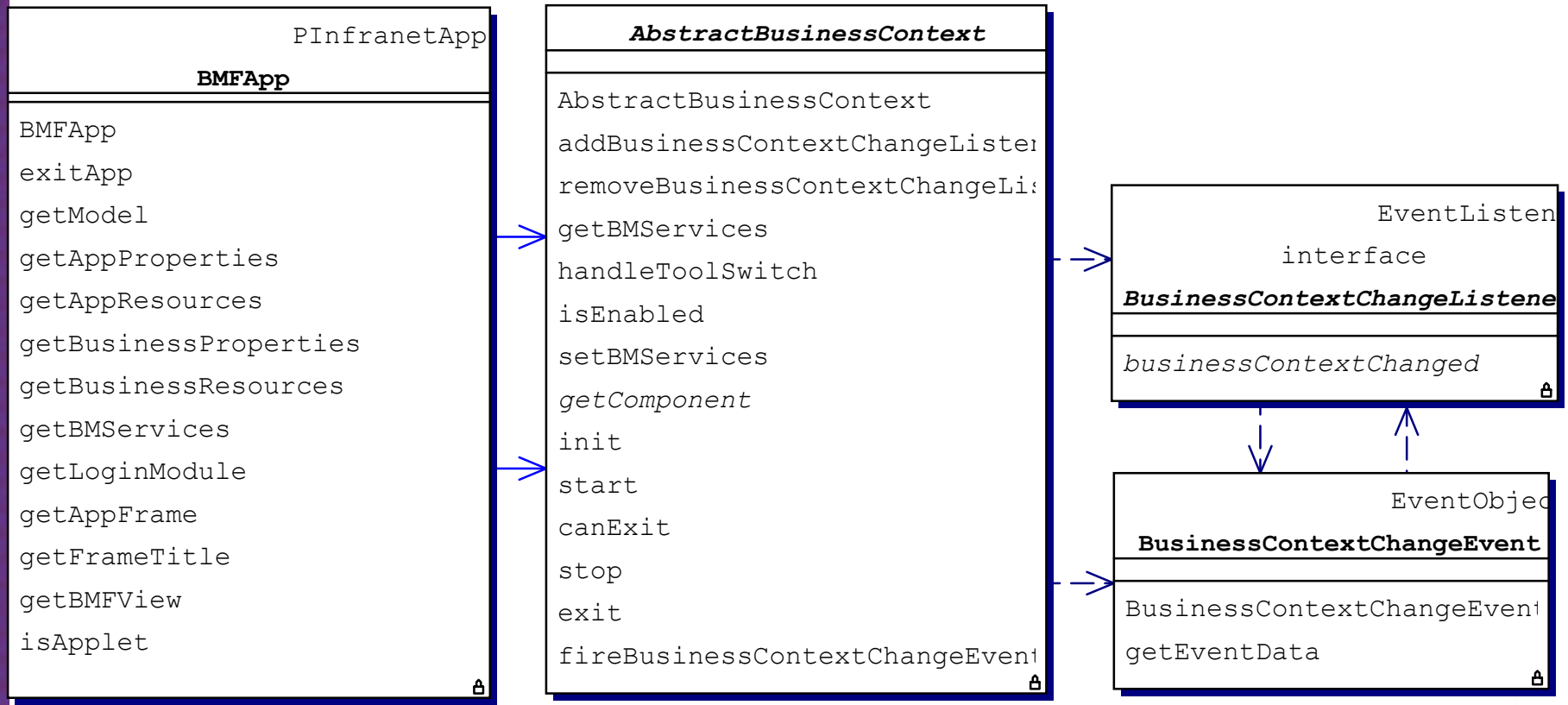


I. Example Implementation Code BMFApp

```
void setActiveTool(AbstractBMFModule newActiveTool) {
    if( newActiveTool == activeTool) {
        return;
    }
    if( activeTool != null) {
        activeTool.setActivated( false);
    }
    activeTool = newActiveTool;
    if( activeTool != null)  {
        activeTool.setActivated( true);
    }
}
```

I. Module Communication

- Context object can be shared between modules
- Can register for notification when it is changed



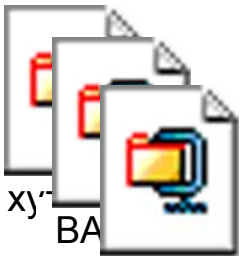
Bad: Monolithic Application Development

Library

Application

Data

Application

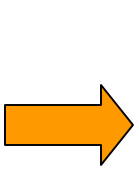


BMF.jar

DevCenter.jar

DevCenterResources.properties

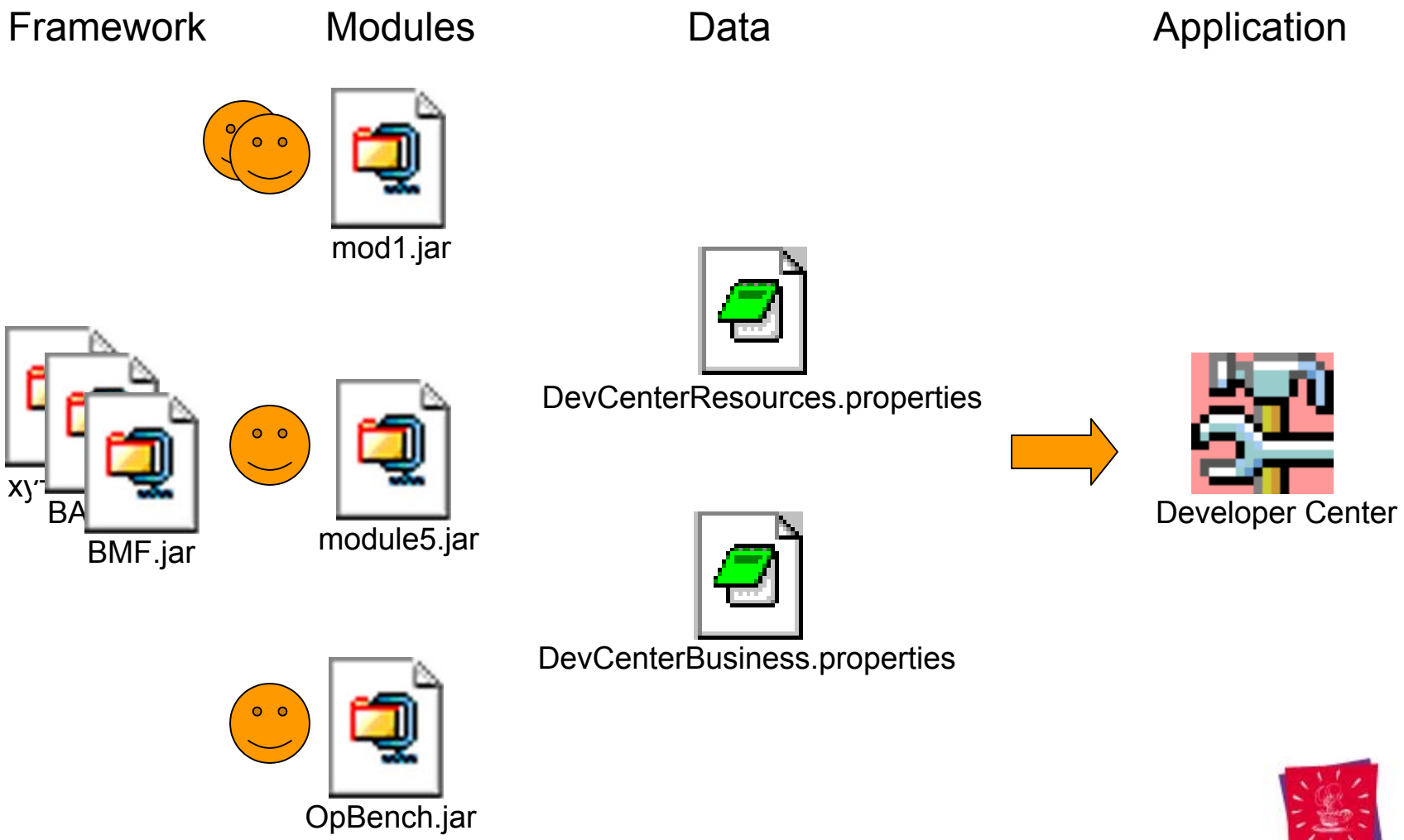
DevCenterBusiness.properties



Developer Center

JavaOne

Good: Modular Development and Applications Assembly



JavaOne

I. User Lessons Learned

- Icon toolbar takes up too much real estate
  - Need better ways for users to display and select from available modules
- Users may need to see more than one module at a time
- Different modules may need to login to different servers

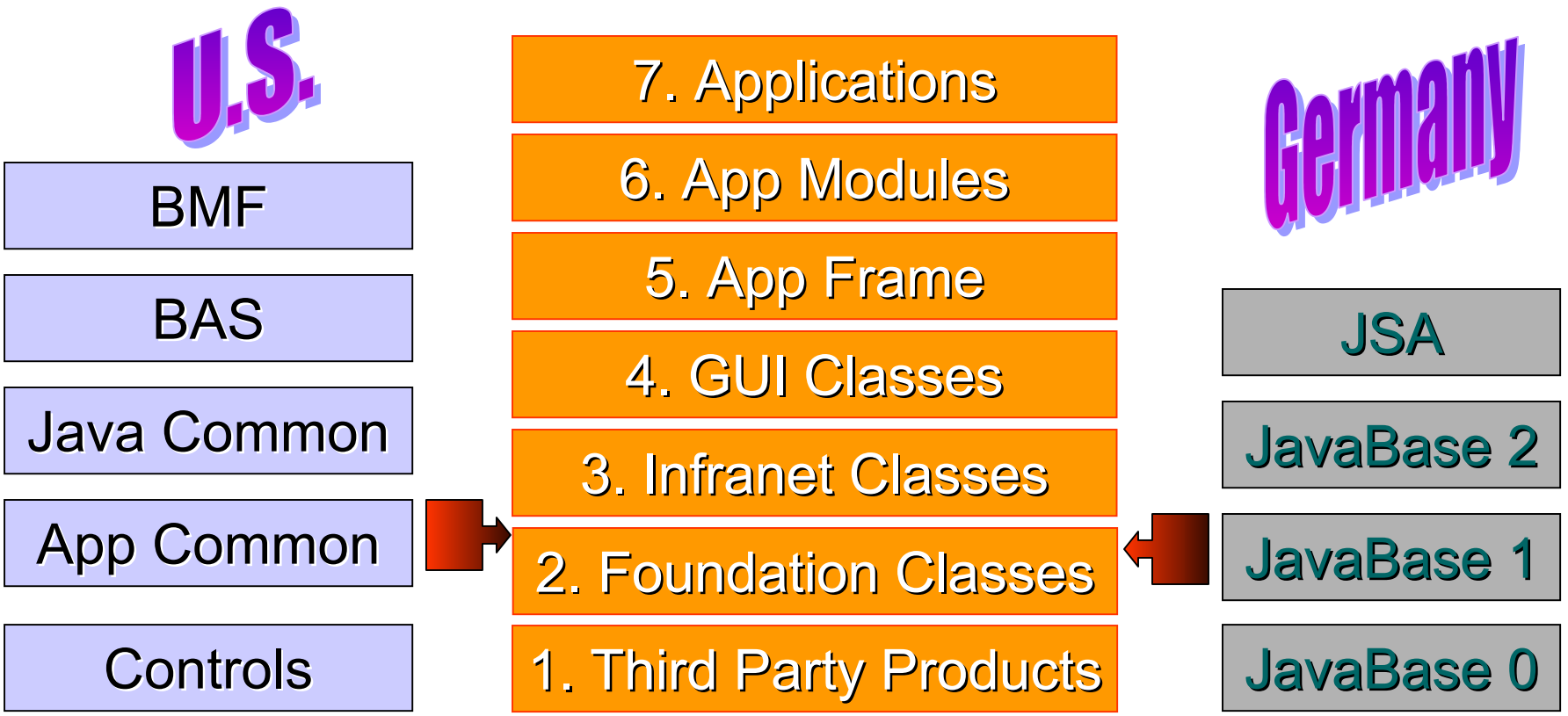
I. Developer Lessons Learned

- Need easy way to keep separate modules synchronized
- Need better way to combine resources from multiple modules
- Be nice to select the modules that users can see based on their names and/or roles
  - Dynamic applications assembly
- Web access important but applets not necessary
  - Use Java WebStart™ instead

II. Portal Framework, for the Java™ Platform: Work In-Progress

- We shipped successful products with the BMF system
- Why design the new Portal Framework system?
  - Build a better foundation with a better architecture
  - Merge existing frameworks and libraries
  - Refactor existing code
  - Fix some limitations
  - Add some new features

II. Portal Framework, for the Java Platform: Merging Existing Libraries



JavaOne

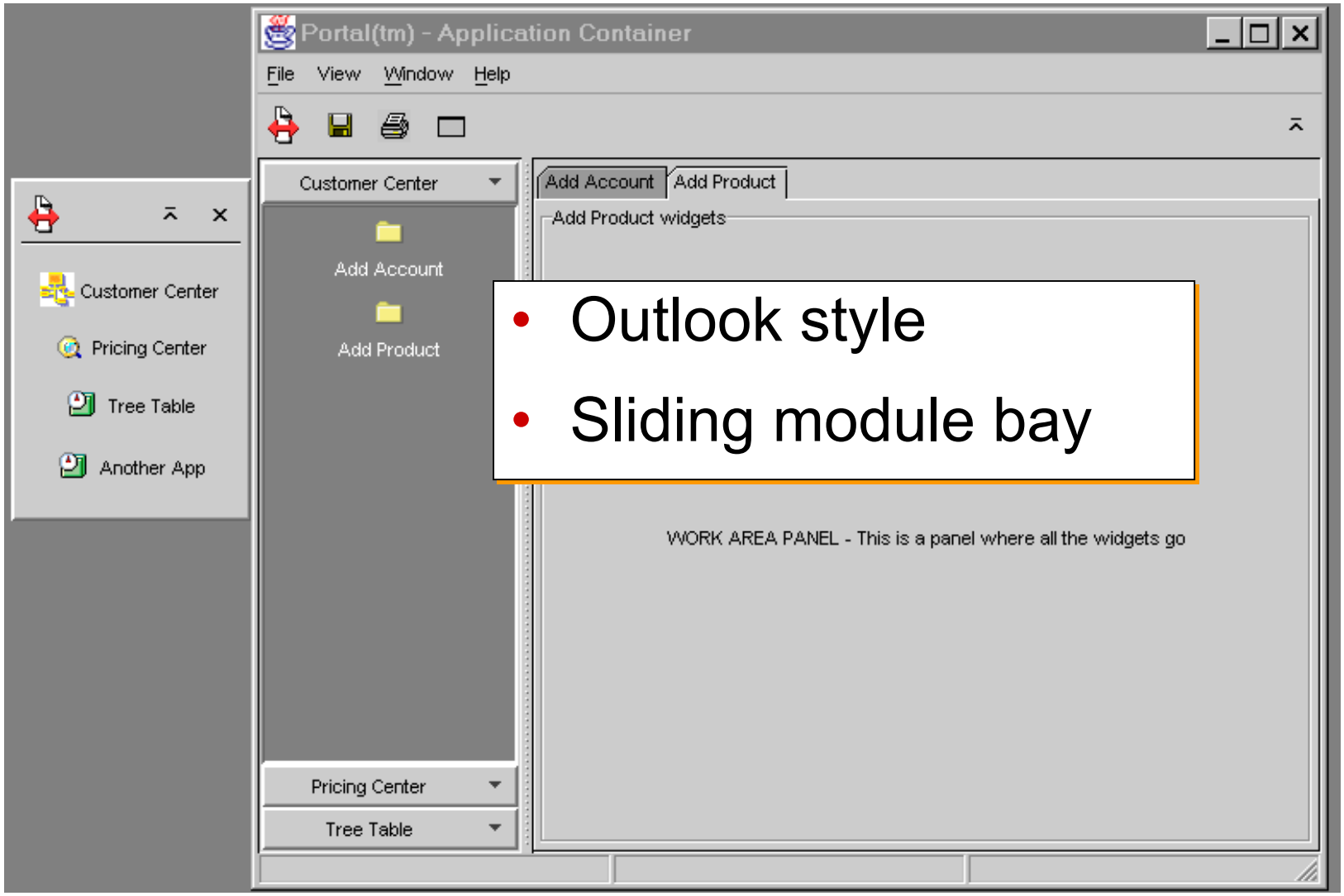
II. Portal Framework, for the Java Platform: Why Seven Layers?

Ii. Portal Framework, for the Java Platform: One-way Dependencies

7. Applications	Apps (Centers)
6. App Modules	Modules
5. App Frame	Container for modules
4. GUI Classes	Widgets (big or small)
3. Infranet Classes	Domain-specific, Non-GUI
2. Foundation Classes	Non-GUI; Non-domain
1. Third-Party Products	Version mgt of JDK™, etc.

- Two-way, cross-dependencies are a maintenance nightmare

II. Portal Framework Prototype

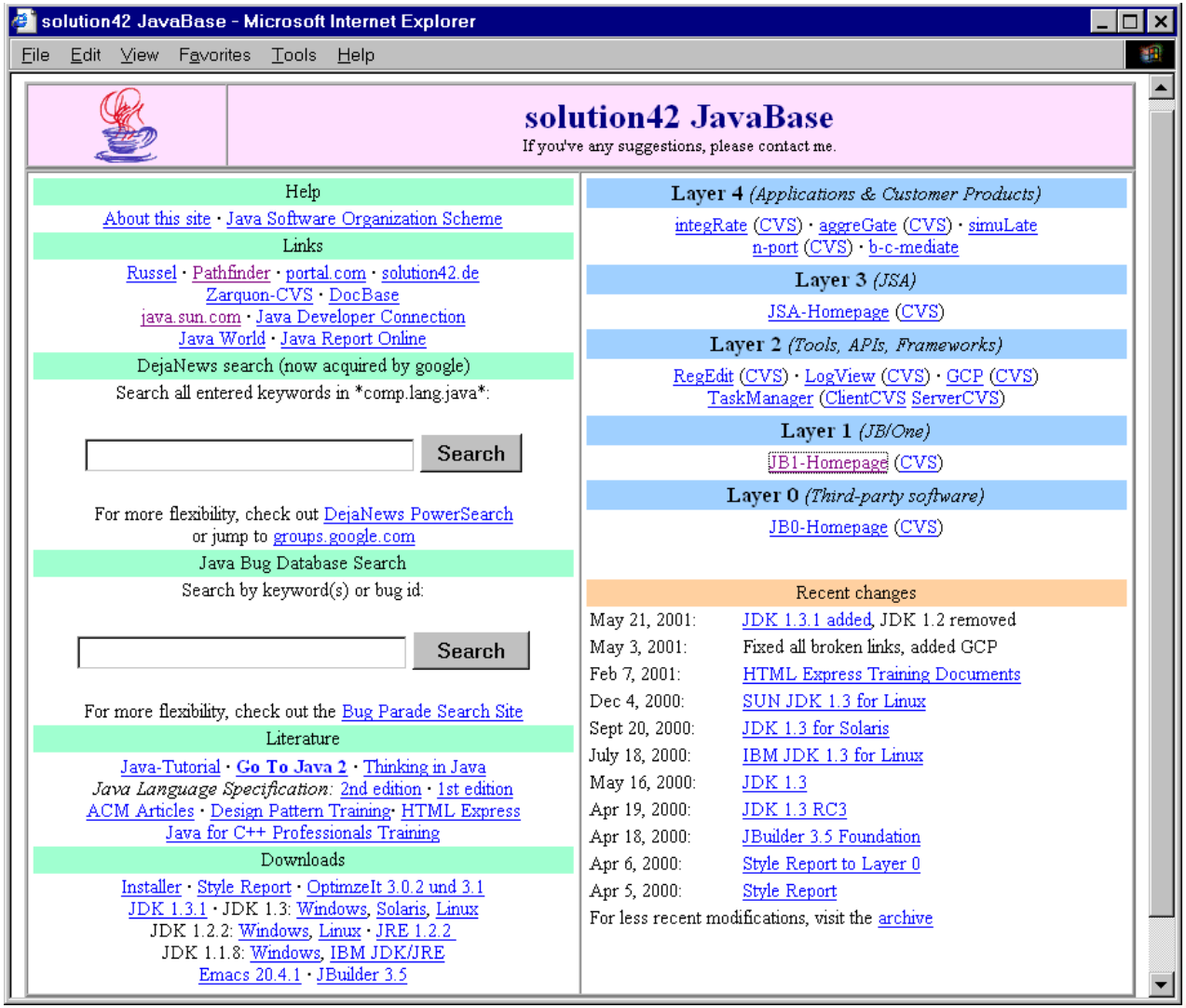


- Outlook style
- Sliding module bay

Demo

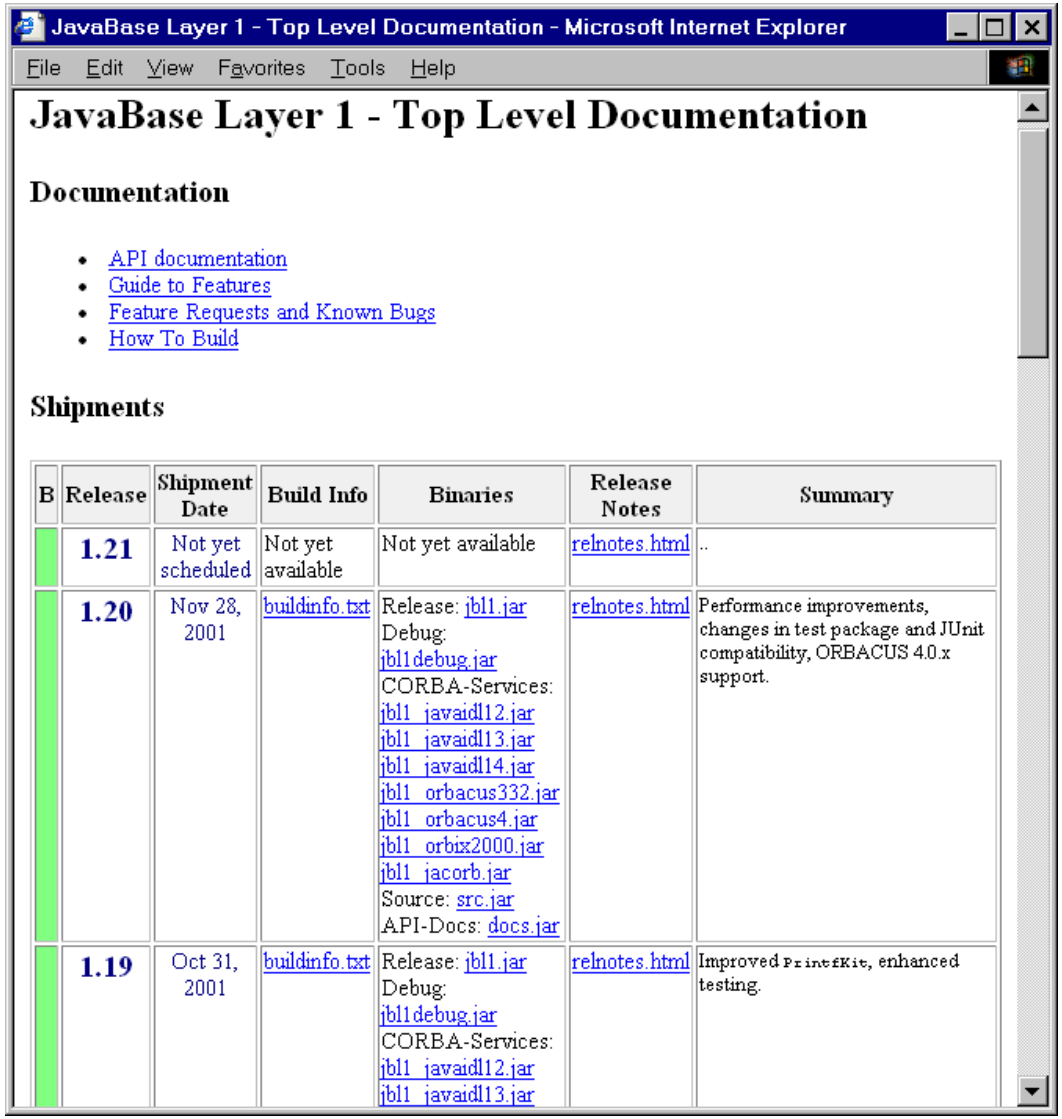
Portal Java Framework
Early Prototype

Documentation



JavaOne

Docs



JavaOne

Summary

- Develop pieces of software as modules
- Use a framework to support combining related modules into application “Centers”
- Make behavior data-driven to minimize the need for subclassing and overriding to change simple behavior
- Minimize abstract methods
  - Provide reasonable default behavior instead
- Provide simple UI mechanisms for users to activate a module, and to switch between modules

If You Only Remember One Thing…

Do **not** build large, monolithic applications
Do **not** build tons of separate small applications
Build functional modules and combine related ones into “centers” for specific activities

Q&A

JavaOne℠

Sun's 2002 Worldwide Java Developer Conference™

BEYOND BOUNDARIES

Session 2689