



# Web Dynpro Componentization

SAP NetWeaver '04


*Bertram Ganz, Christian Georgi, Michael Wenz – SAP AG*

SAP TechEd '05




## Section Overview – Three Building Blocks

1	Component Basics	Component Usage Relations	Demo
2	Component-Based Application Architecture	Hands-On Exercise	
3	NetWeaver Development Infrastructure	Referencing Components Component Interface Definitions	Hands-On Exercise



© SAP AG 2005, SAP TechEd '05 / CD355 / 2


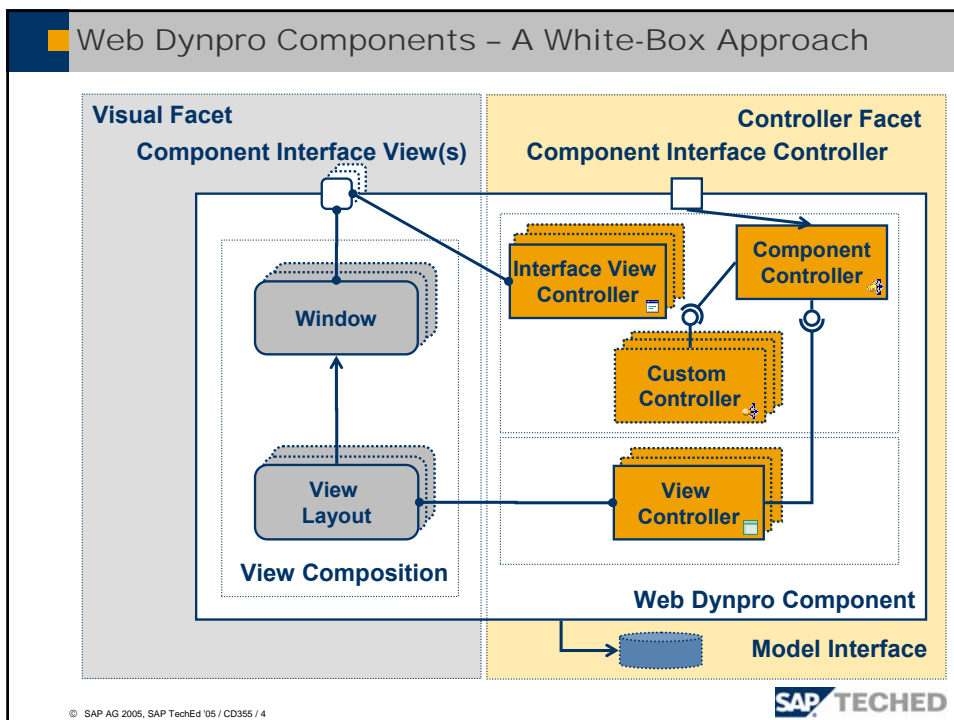
Web Dynpro Component Concept



### Web Dynpro Components ...

- 1 are the fundamental blocks which build Web Dynpro applications
- 2 have a specific controller anatomy (**white-box-approach**)
- 3 have specific interfaces for fulfilling visual, programmatic and context-specific needs (**black-box-approach**)
- 4 are the only unit of reuse in Web Dynpro

© SAP AG 2005, SAP TechEd '05 / CD355 / 3

**Web Dynpro Components – A White-Box Approach**

**Visual Facet**

Component Interface View(s)

Window

View Layout

View Composition

**Visual Facet**

- The Component UI is modeled inside one or many windows
- A **window** contains the **view composition** (views, view sets and the **navigation graph** consisting of linked plugs )
- A **view layouts** contains a tree of Web Dynpro UI-elements
- There is a **1:1 relation** between a window and its component interface view.

© SAP AG 2005, SAP TechEd '05 / CD355 / 5

**SAP** TECHED

**Web Dynpro Components – A White-Box Approach**

**View Layout and View Controller**

- Every View-Layout has a corresponding View Controller
- The View-Controller is there for ...
  - storing and referencing (UI-relevant) context data (data-binding, context mapping)
  - handling user actions
  - triggering navigation steps
  - dynamically modifying the view layout

View Layout

View Composition

1:1

View Controller

**Web Dynpro Component**

© SAP AG 2005, SAP TechEd '05 / CD355 / 6

**SAP** TECHED

Web Dynpro Components – A White-Box Approach

### Non-View Controllers

- Component Controller (1)
- Custom Controllers (0..n)
- Provide global services
- Store global context data
- Eventing
- Context-to-model binding
- Provide public interfaces to other controllers

© SAP AG 2005, SAP TechEd '05 / CD355 / 7

SAP TECHED

Web Dynpro Components – A White-Box Approach

### Controller Interfaces

- **Non-View Controllers** can expose an interface (for accessing the context, calling methods and subscribing to events) to other controllers
- Controllers have to define a **controller usage** relation in order to access the interface of another controller
- View controllers do not expose such an interface to other controllers

© SAP AG 2005, SAP TechEd '05 / CD355 / 8

SAP TECHED

**Web Dynpro Components – A White-Box Approach**

### Interface View Controllers

- Implement event handlers which are called when ..
  - starting (start-up plugs) Web Dynpro applications
  - a component is reached via navigation (inbound plugs)
- Allow firing outbound plugs (navigation)
- Allow firing exit plug
- Have no own context, public methods or events

© SAP AG 2005, SAP TechEd '05 / CD355 / 9

SAP TECHED

**Web Dynpro Components – A White-Box Approach**

### Components and their Model Usages

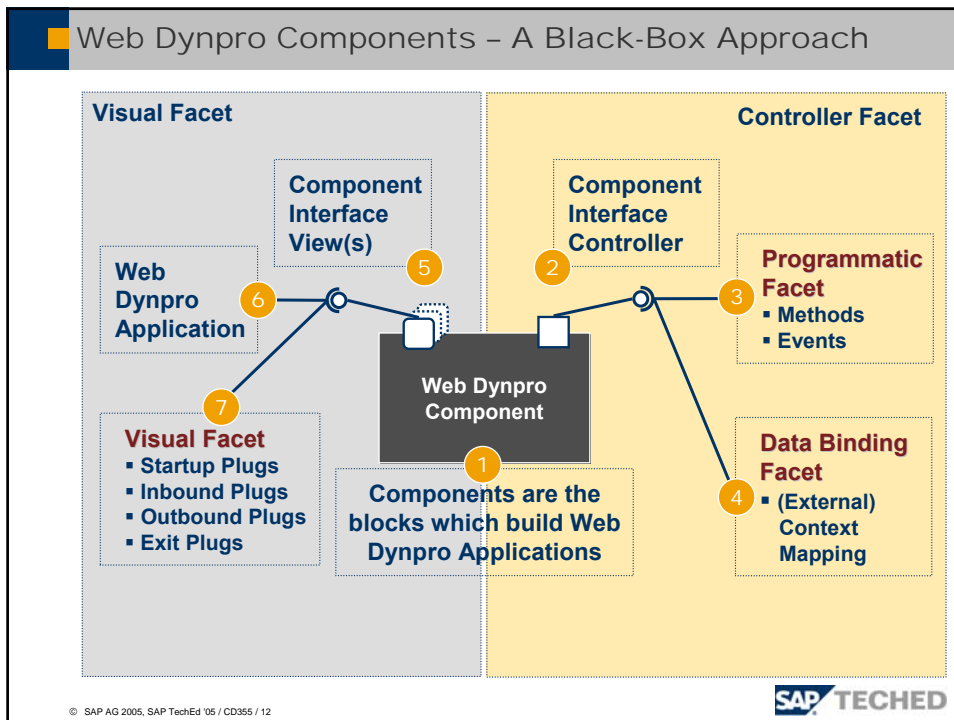
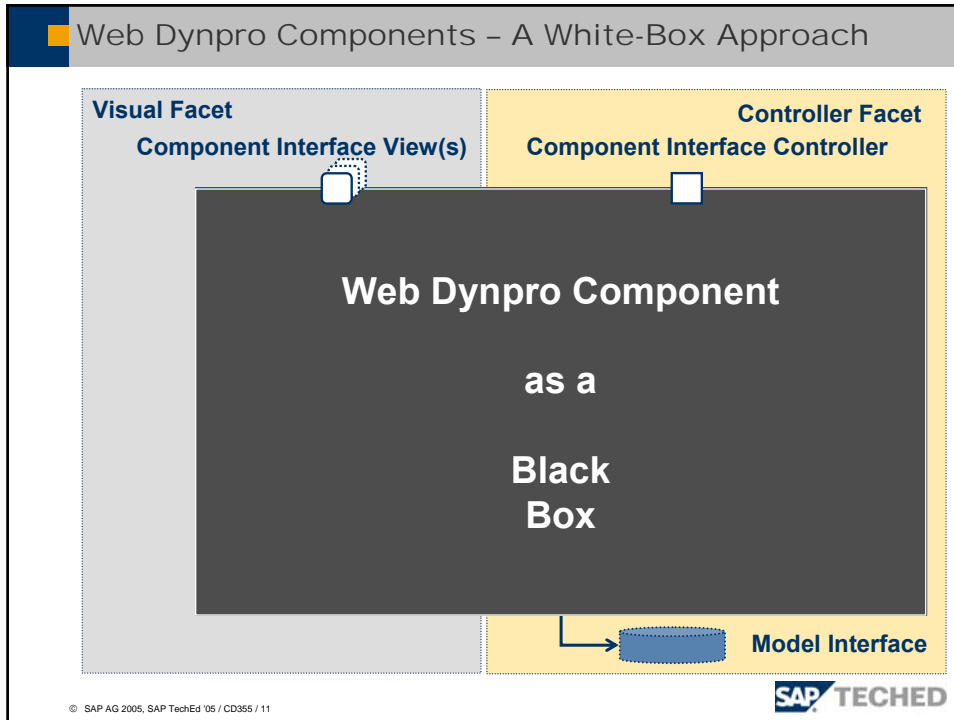
**Proxys** connect to the backend system like mySAP or Web Services

**Model** interfaces provide access to the backend

Data transfer between controller context and model objects is based on a **context-to-model binding** relation

© SAP AG 2005, SAP TechEd '05 / CD355 / 10

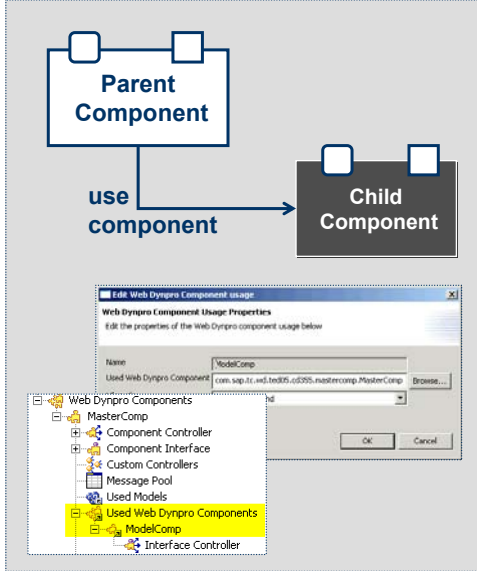
SAP TECHED



### Component Usage Relations

#### Component Usage

- The parent component must explicitly define a **component usage relation** for embedding another child component
- It acts like a **variable** referring to component or component interface definition
- At runtime it is associated with a concrete **component instance**



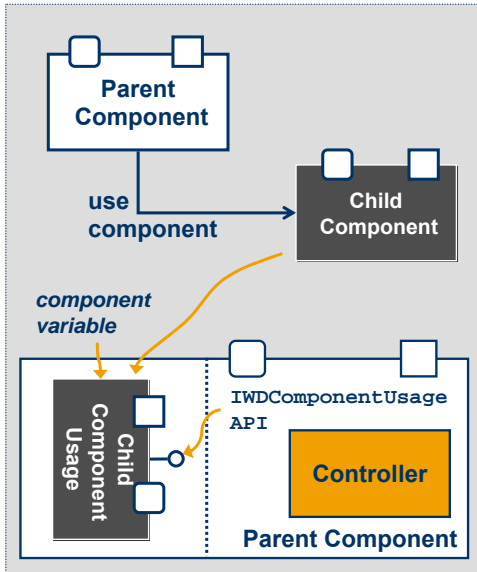
© SAP AG 2005, SAP TechEd '05 / CD355 / 13

SAP TECHED

### Component Usage Relations

#### Component Usage

- Declared inside a Web Dynpro Component (parent)
- Provides two alternative ways of **lifecycle management** for the associated component instance
- Handle to component factory
- Related API: **IWDComponentUsage**



© SAP AG 2005, SAP TechEd '05 / CD355 / 14

SAP TECHED

### Component Usage Relations

#### Component Usage

- Provides access to **instance** of embedded child component
- Enables access to the interface controller
- Enables embedding the component's interface view(s)
  - Optional: e.g. faceless components do not have interface views
  - In the embedding window, interface views behave like "normal" views

© SAP AG 2005, SAP TechEd '05 / CD355 / 15

SAP TECHED

### Managing Component Lifecycles

#### Lifecycle Property of a Component Usage

**createOnDemand:** Web Dynpro Runtime automatically creates a component instance on demand (e. g. when the component gets visible on the UI) and destroys it again

**manual:** component instance lifecycle must be programmatically managed by the the application developer using the **IWDComponentUsage-API**

© SAP AG 2005, SAP TechEd '05 / CD355 / 16

SAP TECHED



### Web Dynpro Component Diagram

**We propose the following diagram type for visualizing Web Dynpro component relations (based on the UML 2.0)**

**Ports of a Web Dynpro Component**

- A Web Dynpro component has special entry points (**ports**) between the component's outer environment and its inner part.
- A port can be seen as the specification of an **interaction point** on the hull of a Web Dynpro component.

**Visual Ports:**  
**Component Interface Views (0..n)**  
 (inbound plugs, outbound plugs, parameters for the inbound plug event handler)

**Service Port:**  
**Component Interface Controller (1)**  
 (context, methods, events)

**SAP** TECHED

© SAP AG 2005, SAP TechEd '05 / CD355 / 17

### Web Dynpro Component Diagram

**Connectors between Web Dynpro Components**

- A connector links a Web Dynpro Component with a port of another component
- The **connector** visualized by linking a **socket symbol** with a **ball symbol**.
- The **socket symbol** starts from the component user
- The **ball symbol** starts from the component interface port of the used component

**Note:**  
*The graphical representation must not explicitly contain the relation dependency (usage relation)*

**Socket notation for a required interface**

**External Interface (Comp. Interf. C):**  
**Ball notation for an exposed interface**


**SAP** TECHED

© SAP AG 2005, SAP TechEd '05 / CD355 / 18

Section 2 – Component Based Application Architecture

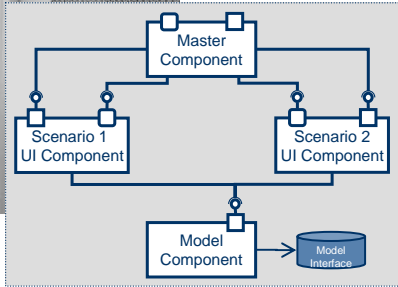

1	Component Basics	Component Usage Relations	Demo
2	Component-Based Application Architecture	Hands-On Exercise	
3	NetWeaver Development Infrastructure	Referencing Components Component Interface Definitions	Hands-On Exercise

© SAP AG 2005, SAP TechEd '05 / CD355 / 19




Breaking Applications into Parts: Components

**Components are the blocks which build Web Dynpro Applications.**



© SAP AG 2005, SAP TechEd '05 / CD355 / 20



Breaking Applications into Parts: Components

Components hide implementation behind an interface (black box approach)

**Benefits**

- Break down applications into manageable parts
  - ◆ Distribute work between developers
  - ◆ Stay independent of implementation details (black box approach)
- Construct reusable entities
  - ◆ A component can be used in several other components

© SAP AG 2005, SAP TechEd '05 / CD355 / 21

SAP TECHED

Breaking Applications into Parts: Components

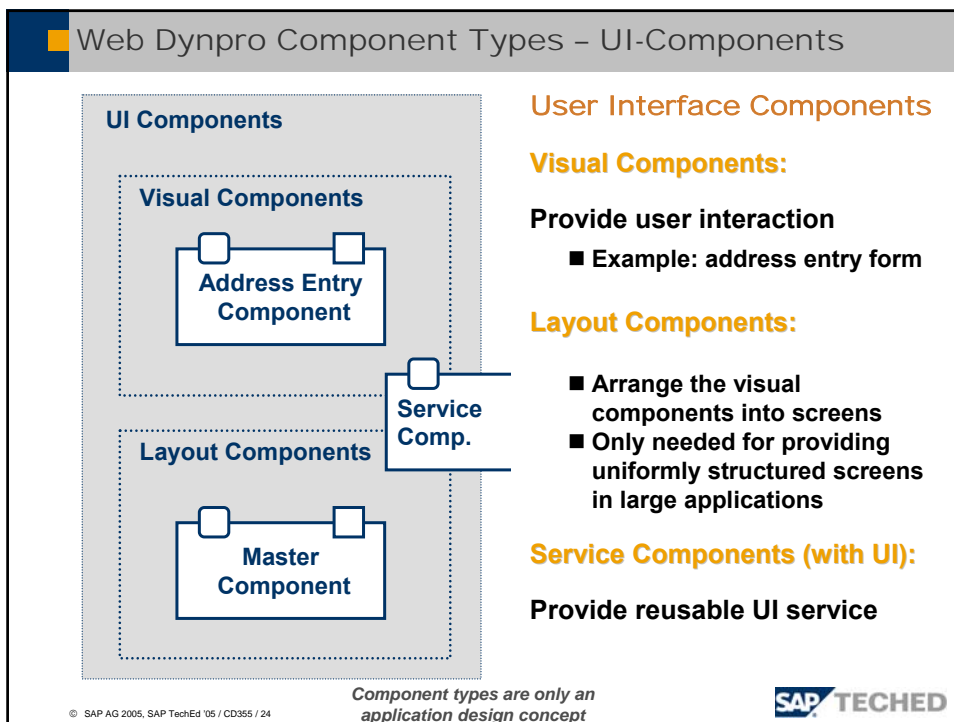
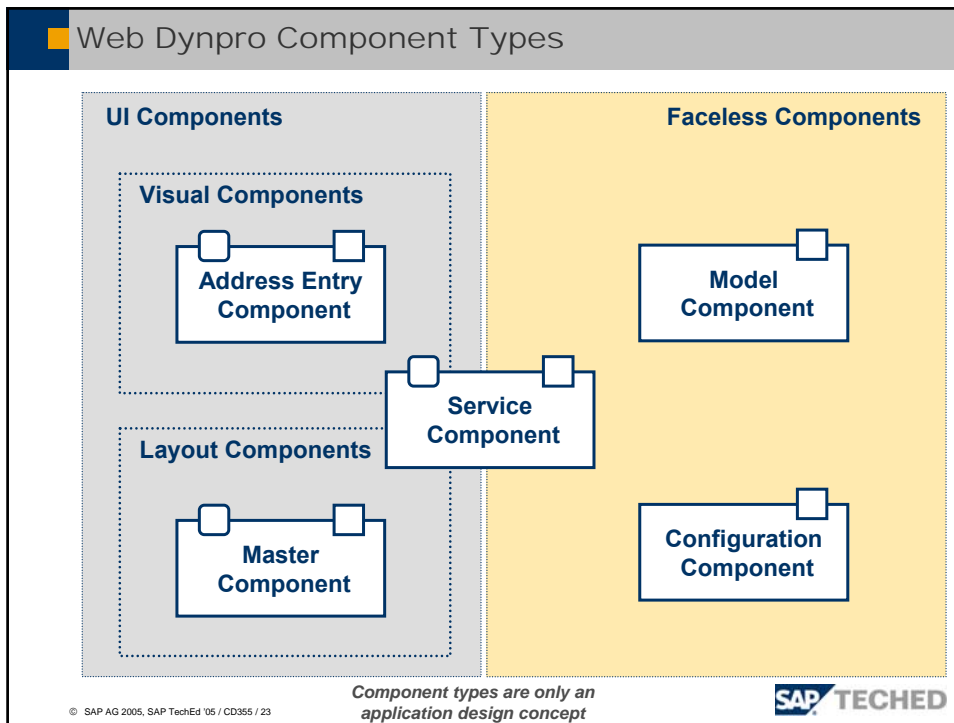
Components concentrate reuse in one element (components, not views ...)

**Benefits**

- Multi-instance capability
  - ◆ A component can embed another one more than once
- Relieve developer from instance management
  - ◆ No multi-instance capability for parts of a component (views, controllers)
  - ◆ Create (non-view) controllers on demand
  - ◆ Get access to controller by specifying its type name

© SAP AG 2005, SAP TechEd '05 / CD355 / 22

SAP TECHED



### Web Dynpro Component Types

**Faceless Component**

**Model Components:**  
Embed models and expose them in their component interface context

**Configuration Components:**  
Special type of model component for storing and retrieving configuration data

**Faceless Components**

*Component types are only an application design concept*

**SAP** TECHED

© SAP AG 2005, SAP TechEd '05 / CD355 / 25

### Components: Component Types

**Use specialized Web Dynpro Component Types**

- User Interface Components
  - ◆ Visual components
  - ◆ Layout components
- Faceless / Model Components
- Service Components

**Component types are only an application design concept**

**Keep UI components and faceless components independent of each other to be able to exchange the other part if this is necessary**

*Component types are only an application design concept*

**SAP** TECHED

© SAP AG 2005, SAP TechEd '05 / CD355 / 26

### Allowing Reuse and Extensibility: Granularity

#### Finding the right component granularity

**1 Too many small components**

the application might not be performant

resulting „sea“ of components might reduce maintainability

**2 Right component granularity**

best reusability, extensibility and maintainability

allows distributing work between several developers

**3 Too large components**

reusability is reduced

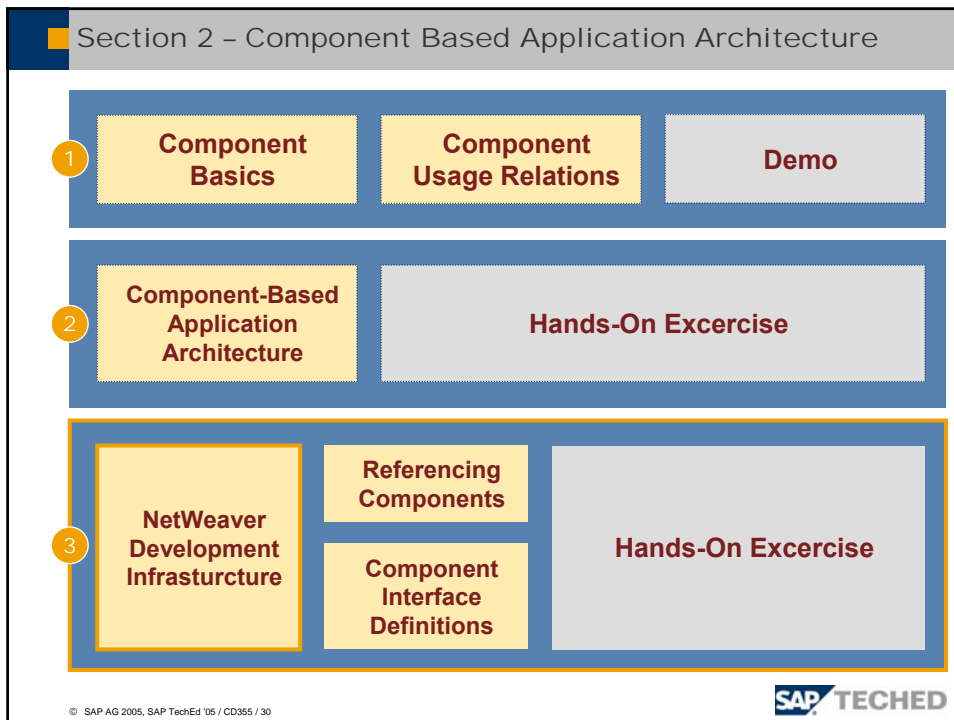
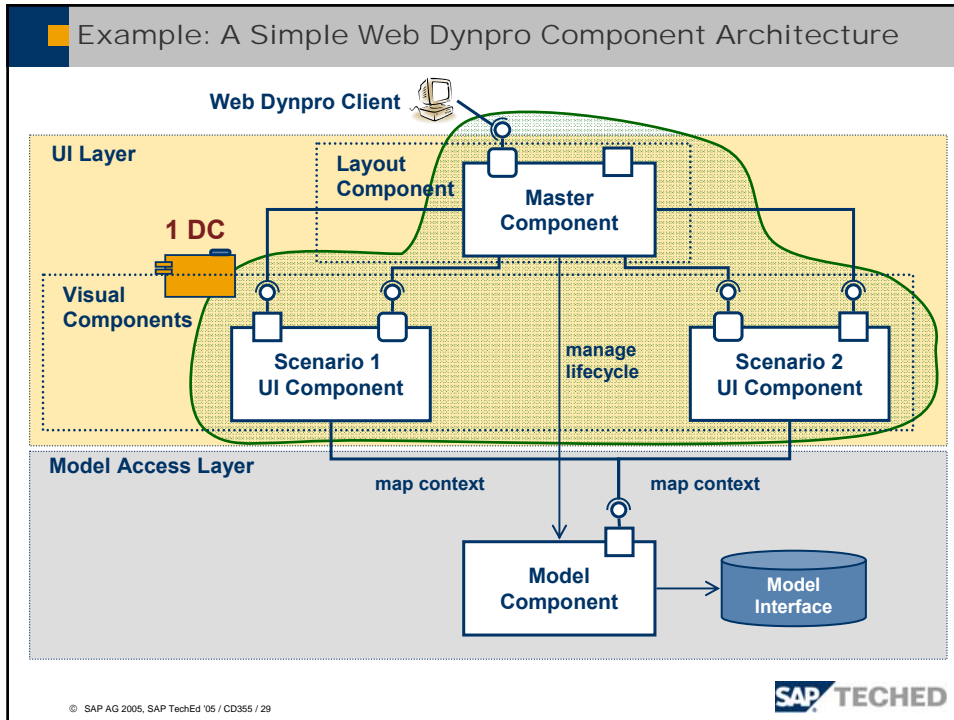
distributing work between several developers is harder

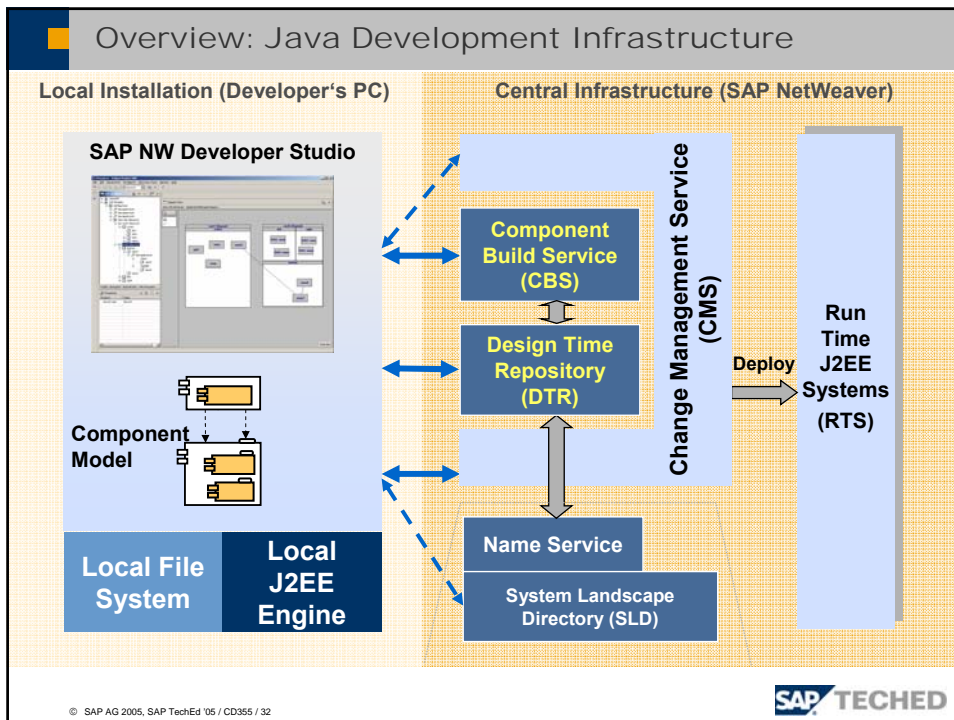
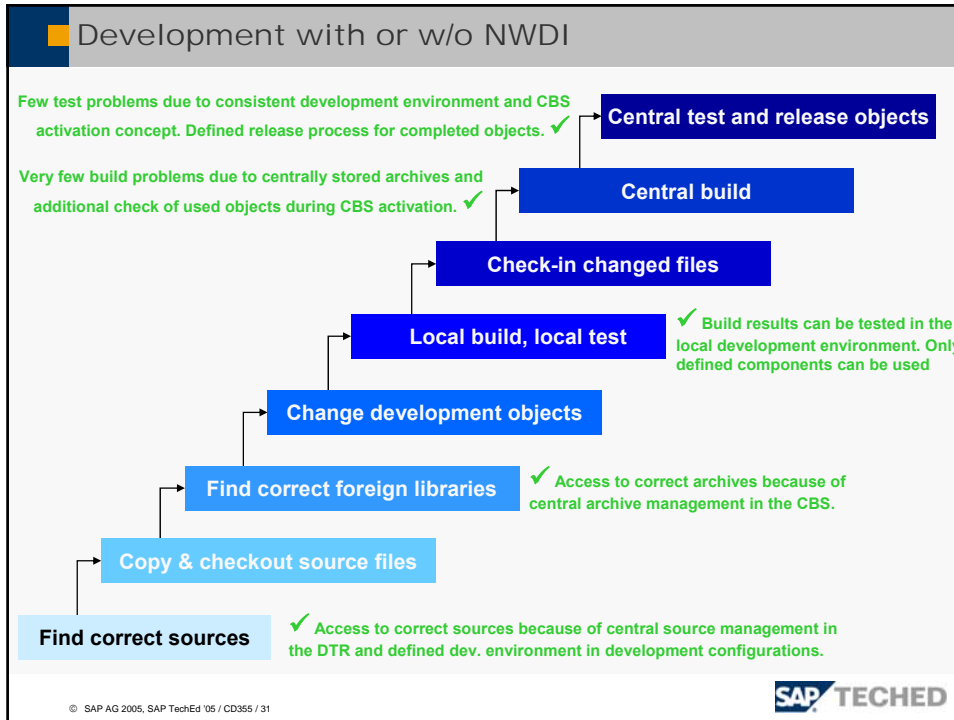
© SAP AG 2005, SAP TechEd '05 / CD355 / 27

### Example: A Simple Web Dynpro Component Architecture

The diagram illustrates a Web Dynpro Component Architecture. At the top, a **Web Dynpro Client** (represented by a computer icon) connects to the **UI Layer**. The UI Layer contains a **Layout Component** and a **Master Component**. Below the UI Layer are **Visual Components**, which include **Scenario 1 UI Component** and **Scenario 2 UI Component**. The **Master Component** is connected to both Scenario UI Components. The **Model Access Layer** contains a **Model Component** and a **Model Interface**. The **Model Component** is connected to the **Model Interface**. The **Scenario 1 UI Component** and **Scenario 2 UI Component** are connected to the **Model Component** via **map context** relationships. The **Master Component** is connected to the **Model Component** via a **manage lifecycle** relationship.

© SAP AG 2005, SAP TechEd '05 / CD355 / 28







## DTR – Basics

### Distributed, concurrent development

- Distributed versioning that spans repositories
- Conflict detection and resolution

### Files and Folders

- DTR knows only files and folders
- No knowledge about objects of programming model (classes, tables, screens etc)

### Check in/check out model

- Files are checked out from DTR and modified “offline”
- Files are checked in again when change is complete

© SAP AG 2005, SAP TechEd '05 / CD355 / 33



## CBS – Features

- Storage of all archives of a software component
- Immediate build of changes on request by developers
- Automatic rebuild of dependent DCs
- Provides access to latest build results for developers
- Web UI (administration)
- Parallel execution of build requests

© SAP AG 2005, SAP TechEd '05 / CD355 / 34



### CBS - Activation Concept

**Inactive Workspace**

- Changes are done only here


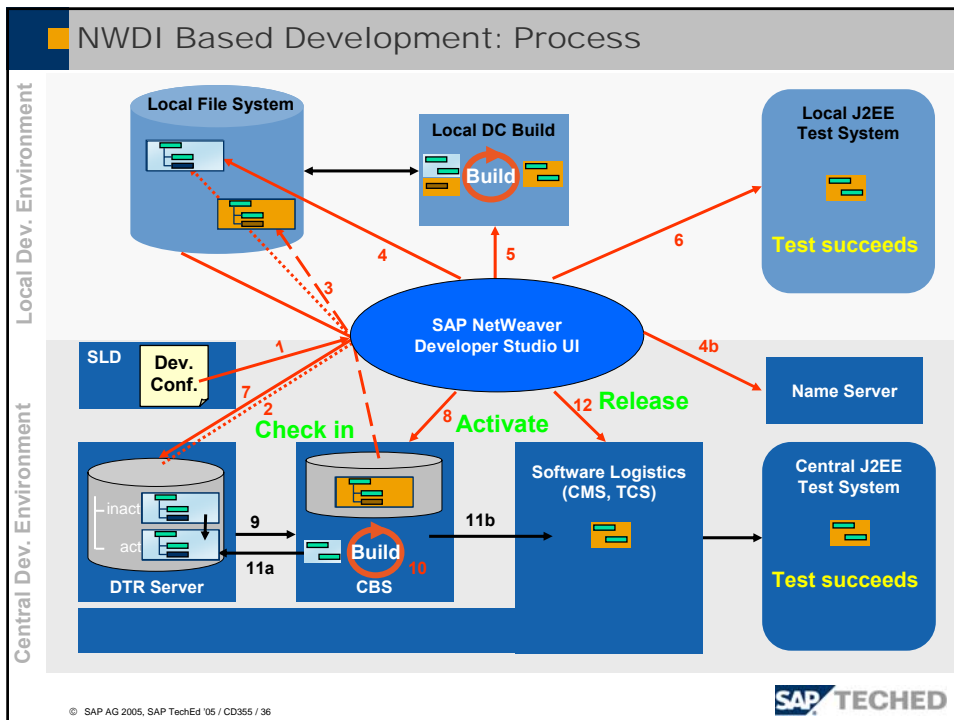
**Active Workspace**

- Has only successfully built sources
- Always in sync with archives available from CBS

**Activating Changes**

- Additional step after check in
- Each code line (DEV, CONS) represented by two DTR workspaces

© SAP AG 2005, SAP TechEd '05 / CD355 / 35

### Structuring Applications Using the Component Model

1. Define a product
2. Define used SCs (in the development configuration)
3. Define new SCs
4. Define top-level DCs
5. Create DCs (default use dependencies to DCs of used SCs are set automatically)
6. Create development objects
7. Create public parts of DCs
8. Set use dependencies

The diagram illustrates the component model structure. At the top, 'Product X' is shown in a dark blue box. Below it, three 'SC' (Software Component) boxes are shown: 'SC A (imported, read only)', 'SC B (imported, read only)', and 'SC X (changeable)'. A dashed blue arrow labeled 'consists of' points from Product X to each SC. Blue arrows labeled 'uses' show SC A using SC B, and SC X using both SC A and SC B. Below each SC, several 'DC' (Development Component) boxes are shown. Green arrows indicate use dependencies between DCs: from SC X to SC A and SC B, and from SC A to SC B.

© SAP AG 2005, SAP TechEd '05 / CD355 / 37

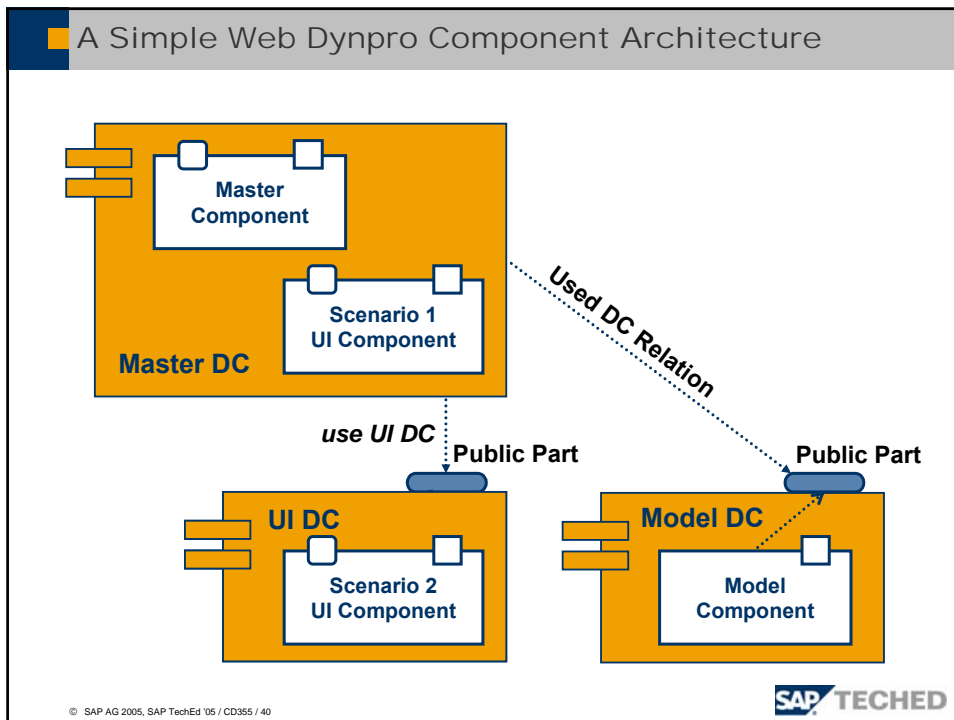
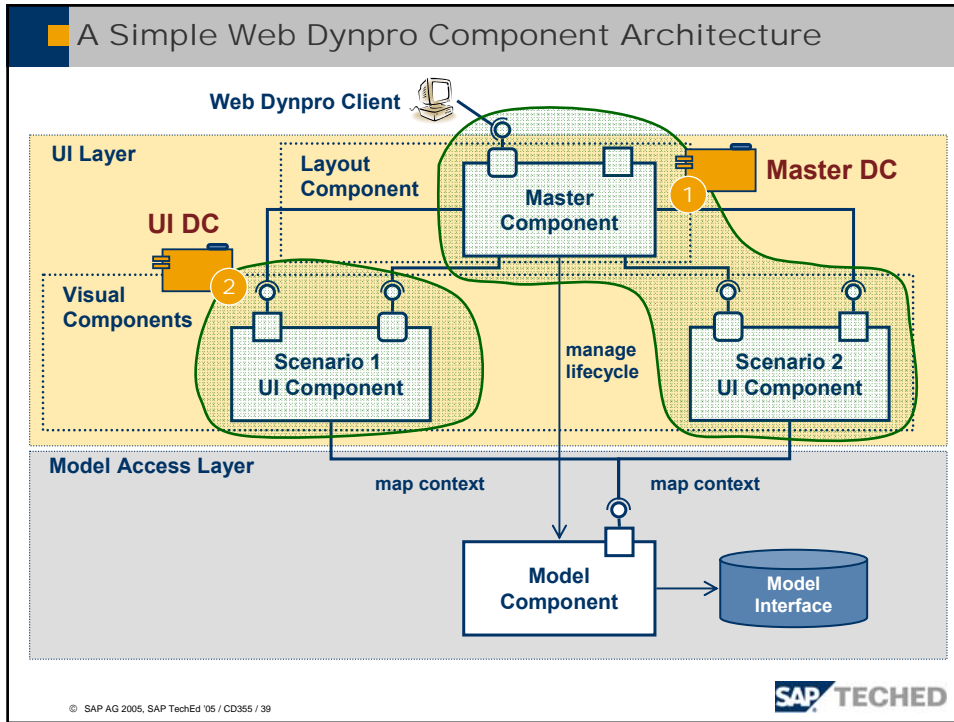
### Allowing Reuse and Extensibility: Delivery

#### Package Web Dynpro Components into Development Components

- A development component is
  - ◆ unit for packaging, building and transportation in SAP's infrastructure (NWDI)
  - ◆ mapped to a project in the NetWeaver Developer Studio
- Distribute objects over development components (reuse, size)
  - ◆ component interface definitions and their implementations
  - ◆ models
- Generally keep development components small
  - ◆ improved build and deploy performance
  - ◆ watch out for increased overhead for maintaining references between them

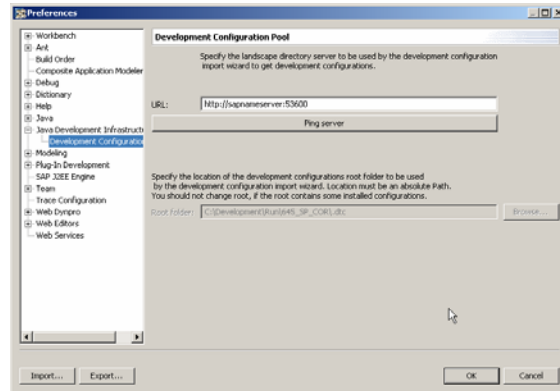
The diagram shows three development components (DC 1, DC 2, DC 3) represented as boxes. DC 1 contains a component labeled '<< Component >> Employee Self Service „Personal Data“' and a component labeled '<< Component Interface Definition >> Detail Display'. DC 2 contains a component labeled '<< Component >> Address Display'. DC 3 contains a component labeled '<< Component >> Bank Account Display'. Arrows point from DC 2 and DC 3 towards the 'Detail Display' interface in DC 1, indicating that these components implement or extend the interface.

© SAP AG 2005, SAP TechEd '05 / CD355 / 38



## SLD Location

- The URL for the system landscape directory has to be entered in the IDE preferences
- Enable IDE to access defined configurations stored within the SLD

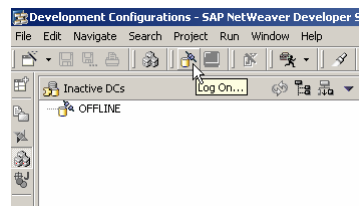


© SAP AG 2005, SAP TechEd '05 / CD355 / 41



## Logon to Development Infrastructure

- Logon needed to work with Java development infrastructure
  - ◆ After every start of the IDE
  - ◆ Preference setting to enable auto logon after IDE start

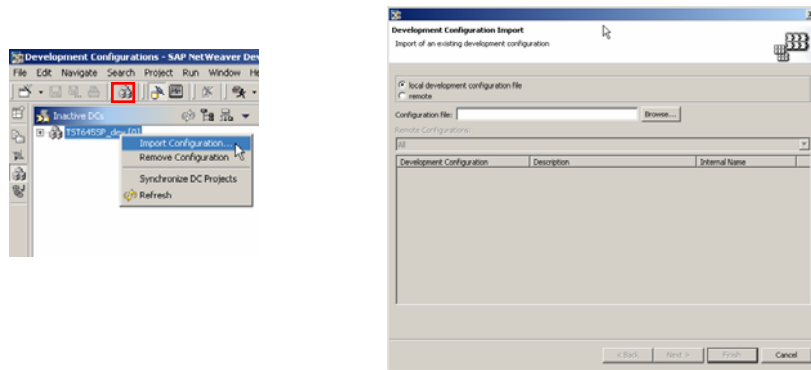


© SAP AG 2005, SAP TechEd '05 / CD355 / 42



## Import configuration

- Start wizard from context menu or toolbar
- Import either
  - ◆ Locally stored configuration file
  - ◆ Configuration from SLD (remote)

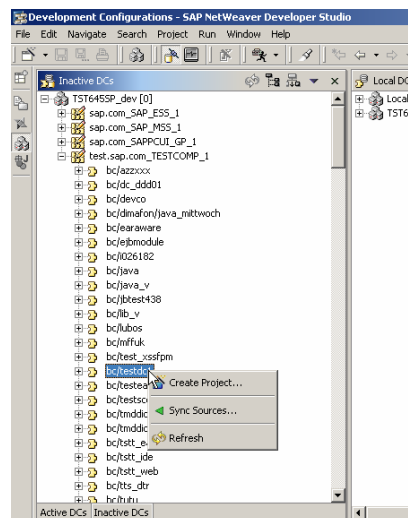


© SAP AG 2005, SAP TechEd '05 / CD355 / 43

SAP TECHED

## Browsing and Syncing Development Components

- Use the development configurations perspective
- Global and local views
  - ◆ Inactive DCs view: global state of inactive DTR workspace
  - ◆ Active DCs view: global state of active DTR workspace
  - ◆ Local DCs view: Local view on available DCs
- Use context menu in the tree to
  - ◆ Sync DCs
  - ◆ Unsync DCs
  - ◆ Create projects for existing DCs to import them into the IDE
    - Do NOT use Eclipse project import for DCs !

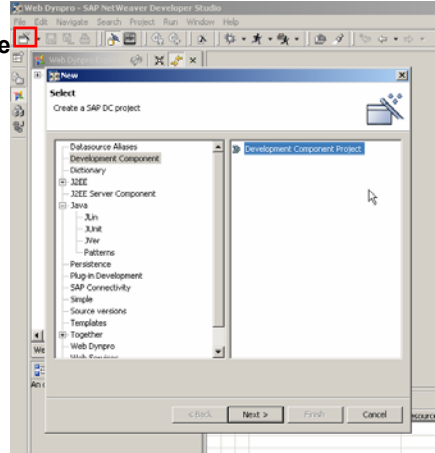
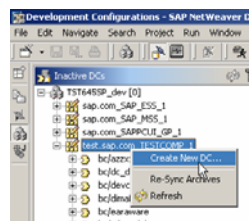


© SAP AG 2005, SAP TechEd '05 / CD355 / 44

SAP TECHED

## Create a new DC

- Create a new DC together with an Eclipse project with the help of the
  - ◆ Eclipse new object wizard
  - ◆ Context menu in the inactive DCs view
- First step: Assign new DC to an existing SC within an imported configuration
- Second step: Name the DC

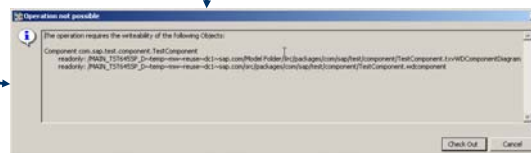
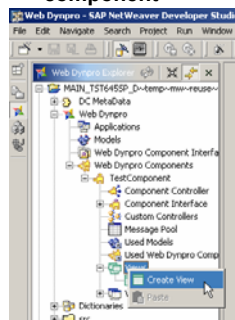
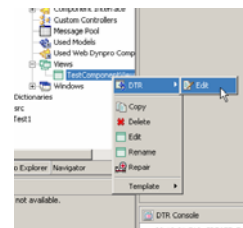


© SAP AG 2005, SAP TechEd '05 / CD355 / 45

SAP TECHED

## Check-out

- Check-out can be explicitly triggered by choosing Edit from the DTR submenu of the context menu of the desired object
- Check-out for Web Dynpro objects will also be implicitly triggered by trying to modify an object
  - ◆ E.g. creating a view within a read-only component will first check-out the component

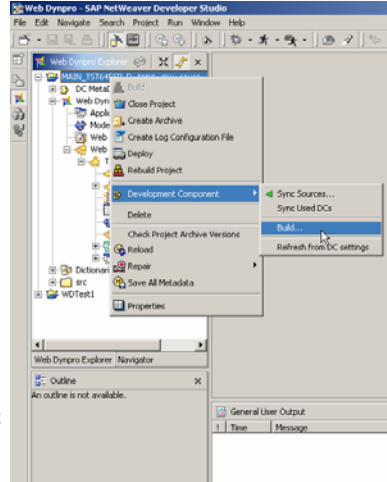


© SAP AG 2005, SAP TechEd '05 / CD355 / 46

SAP TECHED

## Performing DC Build

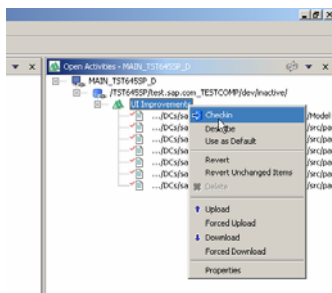
- Start DC build from the context menu of one or more DC projects
  - ◆ Considers the correct build order when called for more than one DC
- Performs
  - ◆ Full build of the DC
  - ◆ Creates a deployable archive
  - ◆ Creates public parts defined by the DC
- Build results are displayed the general user output view
  - ◆ Status message in view
  - ◆ Double click on message to get details
  - ◆ No Eclipse task entries written!
- DC build is necessary
  - ◆ When an object belonging to a public part has been changed
  - ◆ Prior to checking in to assure correctness



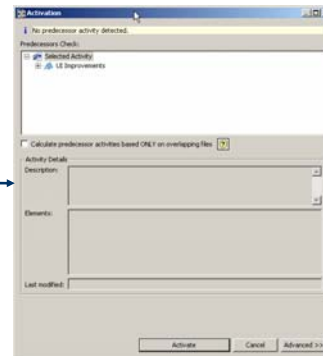
© SAP AG 2005, SAP TechEd '05 / CD355 / 47

SAP TECHED

## Check-in and Activating



- Check-in your changes to DTR by using the context menu entry in the open activities view in the development configuration perspective
- Status and progress messages will appear in general user output view



- After check-in an activation request can be triggered
- Sources and created archives will be moved from inactive to active workspace

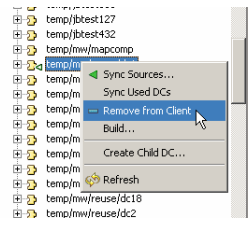
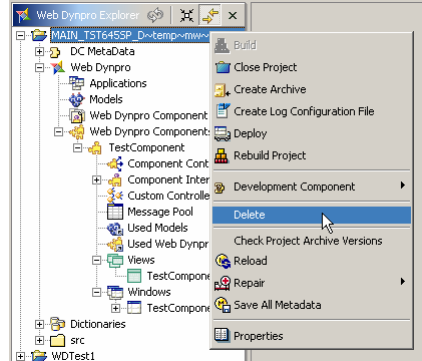
© SAP AG 2005, SAP TechEd '05 / CD355 / 48

SAP TECHED




### Remove DC

- Remove DC by simply deleting the DC project including its files
  - Use delete in context menu of Web Dynpro explorer
  - Use remove from client in local or global DCs view

- Remove DC by simply deleting the DC project including its files
  - Use delete in context menu of Web Dynpro explorer
  - Use remove from client in local or global DCs view
- Local DCs can be deleted completely
- DCs stored in DTR can only be removed from the IDE


© SAP AG 2005, SAP TechEd '05 / CD355 / 49



### Section 2 – Component Based Application Architecture

1	Component Basics	Component Usage Relations	Demo
2	Component-Based Application Architecture	Hands-On Exercise	
3	NetWeaver Development Infrastructure	Referencing Components Component Interface Definitions	Hands-On Exercise

© SAP AG 2005, SAP TechEd '05 / CD355 / 50



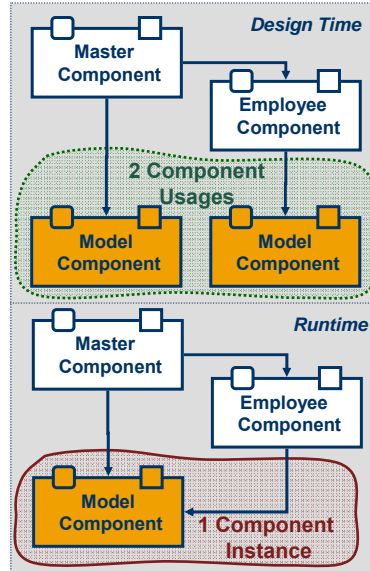
Sharing Component Instance between Component Usages

A **component usage** can be seen as a variable for a Web Dynpro **component instance**.

When using another Web Dynpro component a corresponding **component usage relation** must be declared.

Referencing Mode

- component usages can reference the same component instance
- instance life cycle cannot be controlled in referencing mode
- look on API *IWDCComponentUsage*



© SAP AG 2005, SAP TechEd '05 / CD355 / 51

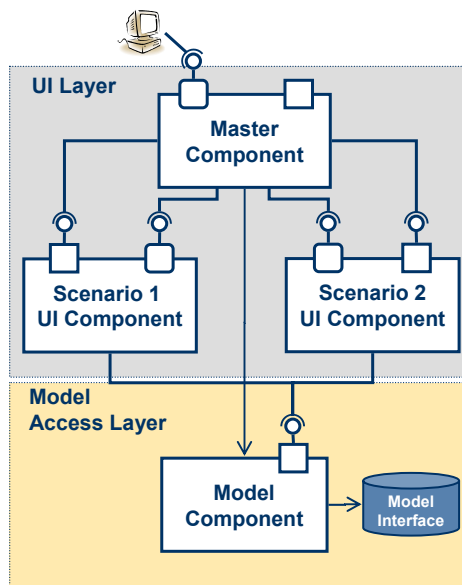


A Simple Web Dynpro Component Architecture

- How can both UI components use the same model component instance?
- But both have their own, independant component usages.

Solution

- The lifecycle of the model component is managed by the master component
- The model component usages in both UI components must define Lifecycle = **manual**.
- The reference to the model component instance is injected by the master component by calling a component interface method of the UI components.
- The UI components can then enter the model component in **referencing mode**.



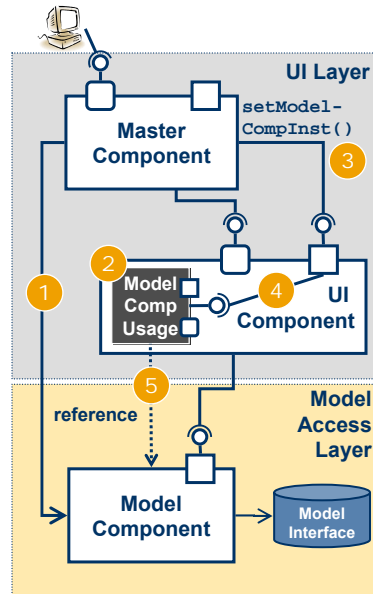
© SAP AG 2005, SAP TechEd '05 / CD355 / 52



Using a Component Instance in Referencing Mode

Entering a Referencing Mode

1. The master component creates an instance of the model component.
2. The UI component has defined a *component usage relation* to the model component. It does not manage the lifecycle (= **manual**) of the used model component itself!
3. The master component passes a reference of the model component usage to the UI component (via a public component interface method).
4. The UI component uses this reference for entering the model component in *referencing mode* (*IWDCComponentUsage-API*).
5. The model component usage is now related to the concrete runtime instance of the model component.



© SAP AG 2005, SAP TechEd '05 / CD355 / 53

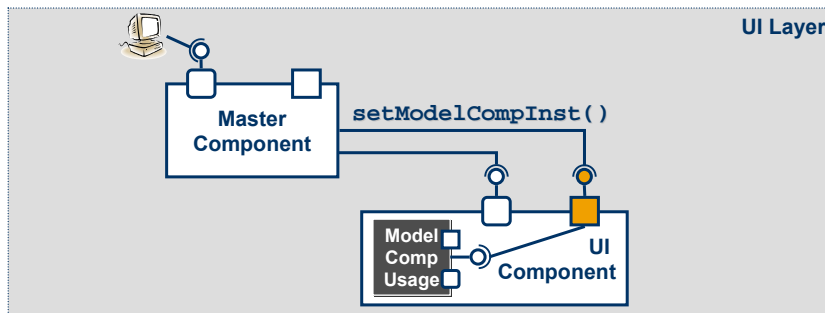


Entering another Component in Referencing Mode

UI Component Interface Controller must expose method

```

setModelComponentInstance(
    IWDCComponentUsage modelCompUsg)
{
    wdThis.wdGet<ModelCompUsageName>
        .enterReferencingMode(modelCompUsg)
}
    
```



© SAP AG 2005, SAP TechEd '05 / CD355 / 54



## Java Interfaces and the Strategy Design Pattern

### Java Interfaces

In Java an interface is a type, just as a class is a type.

Like a class an interface defines methods but never implements them.

An interface is an abstraction of a class independent from its implementation.

```

classDiagram
    class Sale["<<class>> Sale"]
    class IPayment["<<interface>> IPayment"]
    class CreditCardPayment["CreditCard Payment"]
    class CashPayment["Cash Payment"]
    Sale --> IPayment
    IPayment <|-- CreditCardPayment
    IPayment <|-- CashPayment
    
```

Design Time

Runtime

1 <<class>> Sale — <<class>> CreditCard Payment

2 <<class>> Sale — <<class>> Cash Payment

© SAP AG 2005, SAP TechEd '05 / CD355 / 55

SAP TECHED

## Java Interfaces and the Strategy Design Pattern

### The Strategy Design Pattern in Java

*Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.\**

It embodies two main OO design principles:

- *encapsulate the concept that varies*
- *program to an interface, not an implementation.*

**Good** software = loosely coupled collection of interchangeable parts

**Bad** software = monolithic, tightly coupled system

Loose coupling makes your software much more extensible, maintainable, and reusable.

© SAP AG 2005, SAP TechEd '05 / CD355 / 56

\* Chapter 1 of the Gang of Four's (GOF) Design Patterns

SAP TECHED

### Strategy Pattern and Component Interface Definitions

**Component Interface Definitions**

**Allow decoupling Web Dynpro Components**

- Define a component interface without an implementation
- Interface can be implemented by one or more components

**Benefits → Decoupling**

- Component user can program against separately defined component interface (implementation is hidden)
- Component implementation can be provided (or selected) later on (plugin concept)

The diagram illustrates the Strategy Pattern. A **Master Component** depends on a **Component Interface Definition (Suffix Compl)**. Two concrete components, **Address Display** and **Bank Account Display**, implement this interface. The Master Component is responsible for **manage lifecycle** of the interface.

© SAP AG 2005, SAP TechEd '05 / CD355 / 57

**SAP TECHED**

### Example: Adding a Component Interface Definition

The diagram shows a **Web Dynpro Client** architecture. It is divided into two layers:

- UI Layer:** Contains a **Master Component**, **Scenario 1 UI Component**, and **Scenario 2 UI Component**. A **UI DC** (UI Design Component) is associated with the Master Component. The Scenario components implement the **<< WD Compl >> UICompI** interface, which has a `setModelCompInst()` method.
- Model Access Layer:** Contains a **Model Component** and a **Model Interface**. A **Model DC** (Model Design Component) is associated with the Model Component. The Model Component depends on the Model Interface.

Numbered callouts 1, 2, and 3 indicate specific components or relationships.

© SAP AG 2005, SAP TechEd '05 / CD355 / 58

**SAP TECHED**

## Summary

- **Web Dynpro components** are the key to extensible, reusable and manageable Web Dynpro software
- A state-of-the-art Web Dynpro application architecture can be achieved by using **specialized component types**
- The **NetWeaver Development Infrastructure** provides functions for developing professional, component-based Web Dynpro applications

© SAP AG 2005, SAP TechEd '05 / CD355 / 59



## Further Information

### → Public Web:

[www.sap.com](http://www.sap.com)

SAP Developer Network: [www.sdn.sap.com](http://www.sdn.sap.com) → Web Application Server → Web Dynpro (→ Code Samples → Sample Applications and Tutorials)

SAP Customer Services Network: [www.sap.com/services/](http://www.sap.com/services/)

### → Related SAP Education Training Opportunities

<http://www.sap.com/education/>

### → Related Workshops/Lectures at SAP TechEd 2005

CD210, Transaction Handling in Web Dynpro, 1-Hour Lecture

CD213, Web Dynpro Meets JSF – A New Level of Interoperability , 1-H L.

CD258 Putting It All Together –

Application Development in SAP NetWeaver, 4-H Workshop

CD255 Component-Based Java Development with SAP NetWeaver, 4-H W.

CD207 SAP NetWeaver Development Infrastructure – An Overview, 1-H L.

© SAP AG 2005, SAP TechEd '05 / CD355 / 60



## Copyright 2005 SAP AG. All Rights Reserved

- No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.
  - Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.
  - Microsoft, Windows, Outlook, and PowerPoint are registered trademarks of Microsoft Corporation.
  - IBM, DB2, DB2 Universal Database, OS/2, Parallel Sysplex, MVS/ESA, AIX, S/390, AS/400, OS/390, OS/400, iSeries, pSeries, xSeries, zSeries, z/OS, AFP, Intelligent Miner, WebSphere, Netfinity, Tivoli, and Informix are trademarks or registered trademarks of IBM Corporation.
  - Oracle is a registered trademark of Oracle Corporation.
  - UNIX, X/Open, OSF/1, and Motif are registered trademarks of the Open Group.
  - Citrix, ICA, Program Neighborhood, MetaFrame, WinFrame, VideoFrame, and MultiWin are trademarks or registered trademarks of Citrix Systems, Inc.
  - HTML, XML, XHTML and W3C are trademarks or registered trademarks of W3C®, World Wide Web Consortium, Massachusetts Institute of Technology.
  - Java is a registered trademark of Sun Microsystems, Inc.
  - JavaScript is a registered trademark of Sun Microsystems, Inc., used under license for technology invented and implemented by Netscape.
  - MaxDB is a trademark of MySQL AB, Sweden.
  - SAP, R/3, mySAP, mySAP.com, xApps, xApp, SAP NetWeaver, and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world. All other product and service names mentioned are the trademarks of their respective companies. Data contained in this document serves informational purposes only. National product specifications may vary.
- 
- The information in this document is proprietary to SAP. No part of this document may be reproduced, copied, or transmitted in any form or for any purpose without the express prior written permission of SAP AG.
  - This document is a preliminary version and not subject to your license agreement or any other agreement with SAP. This document contains only intended strategies, developments, and functionalities of the SAP® product and is not intended to be binding upon SAP to any particular course of business, product strategy, and/or development. Please note that this document is subject to change and may be changed by SAP at any time without notice.
  - SAP assumes no responsibility for errors or omissions in this document. SAP does not warrant the accuracy or completeness of the information, text, graphics, links, or other items contained within this material. This document is provided without a warranty of any kind, either express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, or non-infringement.
  - SAP shall have no liability for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of these materials. This limitation shall not apply in cases of intent or gross negligence.
  - The statutory liability for personal injury and defective products is not affected. SAP has no control over the information that you may access through the use of hot links contained in these materials and does not endorse your use of third-party Web pages nor provide any warranty whatsoever relating to third-party Web pages.