



DEV399 Essentials of IBM Rational Software Modeler for Systems Development

Student Guide

Part No. 800-027180-000

IBM Corporation
Rational University
DEV399 Essentials of IBM Rational Software Modeler for Systems Development
Student Guide

December 2005

Copyright © International Business Machines Corporation, 2005. All rights reserved.

This document may not be reproduced in whole or in part without the prior written permission of IBM.

The contents of this manual and the associated software are the property of Rational Software and/or its licensors, and are protected by United States copyright laws, patent laws, and various international treaties. For additional copies of this manual or software, please contact Rational Software. IBM and the IBM logo are trademarks or registered trademarks of IBM Corporation, in the United States, other countries or both.

Rational, the Rational logo, ClearCase, ClearCase LT, ClearCase MultiSite, Unified Change Management, Rational SoDA, and Rational XDE are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries or both.

WebSphere, the WebSphere logo, and Studio Application Developer, are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries or both.

Microsoft Windows 2000, Microsoft Word, and Internet Explorer, among others, are trademarks or registered trademarks of Microsoft Corporation.

Java and all Java-based marks, among others, are trademarks or registered trademarks of Sun Microsystems in the United States, other countries or both.

UNIX is a registered trademark of The Open Group in the United States, other countries or both.

Other company, product and service names may be trademarks or service marks of others.

Printed in the United States of America.

This manual prepared by:
IBM Rational Software
18880 Homestead Road
Cupertino, CA 95014-0721
USA

Contents

Module 0 About This Course

Intended Audience	0-3
Course Goals and Objectives	0-5

Module 1 Getting Started

Introduction to Rational Software Modeler	1-3
The Modeling Perspective	1-7
Views in the Modeling Perspective	1-10
Rational Software Modeling Projects	1-19

Module 2 Model Structure and Templates

Model Templates in Rational Software Modeler	2-3
Demo and Lab	2-9

Module 3 Creating UML Diagrams

UML Diagrams	3-3
Drawing UML Diagrams.....	3-18

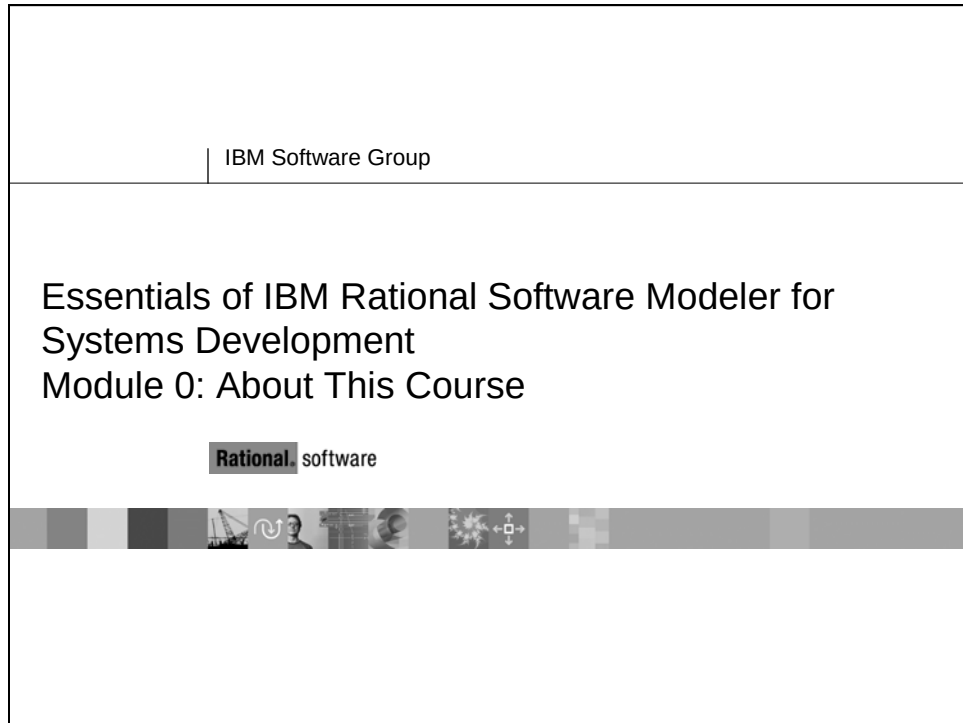
Module 4 UML Diagrams for Systems Modeling

Context Diagrams	4-3
Sequence Diagrams for Use-Case Flowdown.....	4-11
Locality Diagrams	4-15

Module 5 Team Development

Configuration Management in Rational Software Modeler	5-4
Compare and Merge.....	5-9
Model Publishing.....	5-15

► ► ► **Module 0**
About This Course



Topics

Intended Audience.....	0-3
Course Goals and Objectives.....	0-5

Introductions

Introductions

- ◆ Your organization
- ◆ Your role
- ◆ Your background and experience
 - Software development experience
 - Object technology experience
- ◆ Course expectations



Intended Audience

Intended Audience

- ♦ System engineers and system and software architects who need to:
 - Work with requirements.
 - Model systems.
 - Enforce architectural standards.
- ♦ Software engineers who need to:
 - Develop using models and code.
 - Review code against coding standards.
 - Apply patterns and transformations.
- ♦ Everyone who develops as part of a team.



3

IBM

Systems engineers are responsible for specifying requirements and architecture for systems that include hardware and software components and external interfaces to the system, including user interaction. They are responsible for ensuring that the system as built meets the requirements; for validation and verification, as well as testing of the overall combined system.

Software architects and senior developers within a development team are responsible for specifying and maintaining all aspects of an application's software architecture. They need powerful and configurable tools for managing the complexity found in today's applications.

IBM Rational Modeling Tools are UML model-driven design and development tools for creating well-architected systems, applications and services.

Essentials of IBM Rational Software Modeler for Systems Development is for system engineers, system architects, software architects, and engineers who create systems using the Rational Unified Process and Model-Driven Development (MDD) or Model-Driven Architecture (MDA) or similar processes for the development of systems.

Prerequisites

Prerequisites

This course assumes that students have knowledge of:

- Rational Unified Process for Systems Engineering
- Applying object-oriented concepts and UML to system problems

Course Goals and Objectives

Course Goals and Objectives

After completing this course, you will be able to perform the following tasks with Rational Software Modeler:

- Describe its basic features.
- Use Rational Software Modeler template models and model building blocks to reduce repetitive labor.
- Identify the UML diagrams available in Rational Software Modeler.
- Draw system-related diagrams to model systems.
- Describe team development best practices.

5



This course teaches system engineers, system and software architects, software developers and all who develop as part of a team how to apply Rational Software Modeler and the Rational Unified Process (RUP).

Course Outline

Course Outline	
Morning:	
About This Course	½ hour
Getting Started	1 hour
Model Structure and Templates	1 ½ hour
Creating UML Diagrams	2 hours
Lunch	
	1 hour
Afternoon:	
UML Diagrams for Systems Modeling	2 hours
Team Development	1 hour

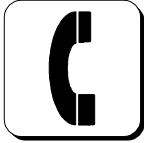
6

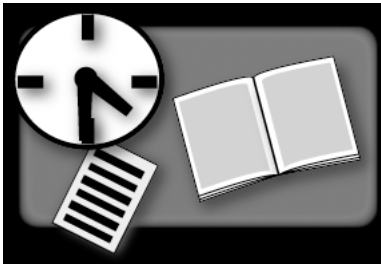


Times for modules are estimates only. Class times will vary.

Logistics

Logistics





Morning
2 Fifteen-minute breaks

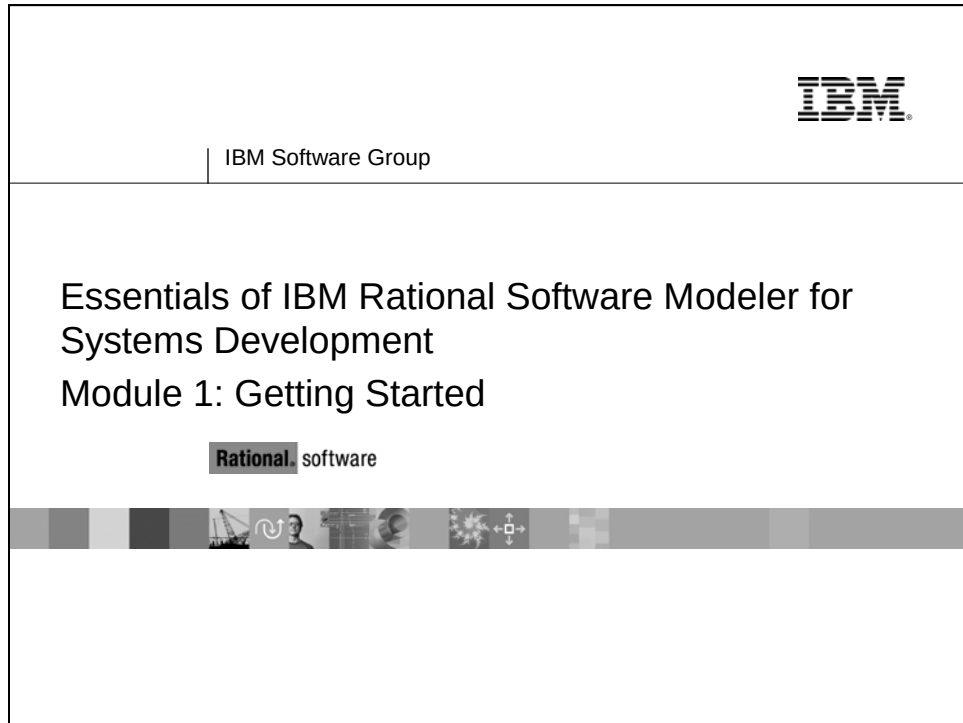
Lunch
1 Hour

Afternoon
2 Fifteen-minute breaks

7



► ► ► **Module 1**
Getting Started



Topics

Introduction to Rational Software Modeler.....	1-3
The Modeling Perspective	1-7
Views in the Modeling Perspective.....	1-10
Rational Software Modeler Projects	1-19

Getting Started

Getting Started

Objectives:

- Describe the basic features of Rational Software Modeler.
 - Describe the **Modeling** perspective and its views and editors.
 - Describe the project types available.
- Create a simple UML model.
 - Create a modeling project and add a new model from a model template.
 - Create, populate, and validate a class diagram.

2



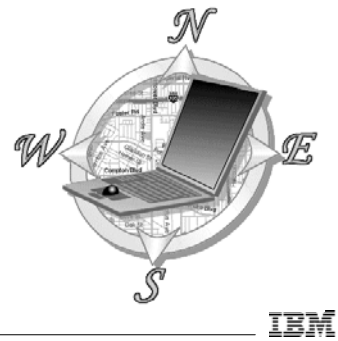
This module describes the Rational Software Modeler workbench's **Modeling** perspective and the unique views it features.

It also presents an overview of the Software Modeler project and model types.

Introduction to Rational Software Modeler

Where Are We?

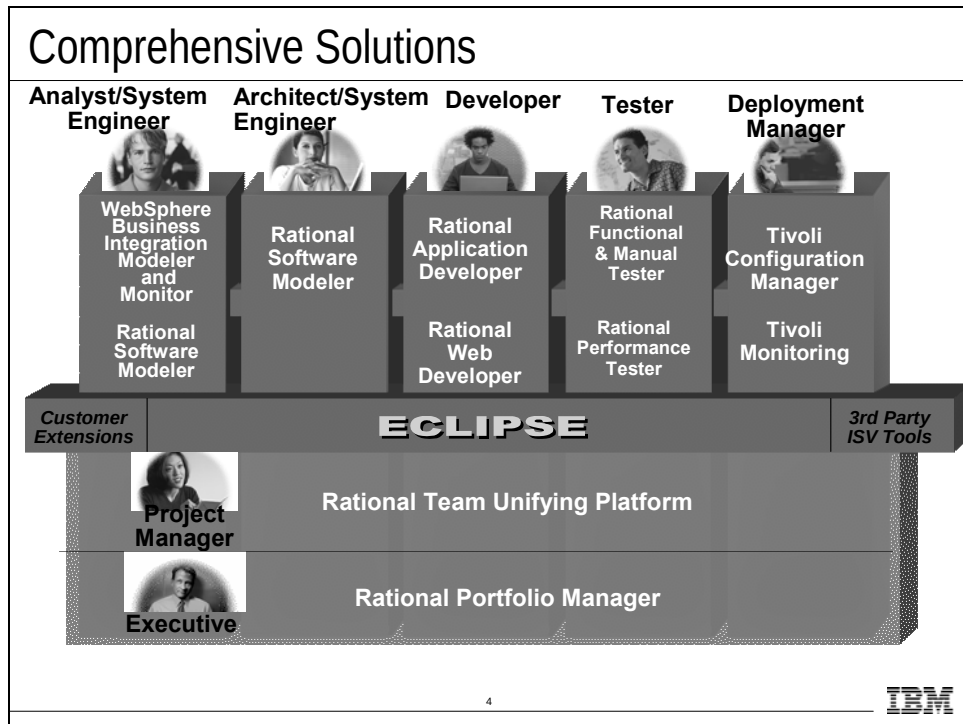
- ♦ **Introduction to Rational Software Modeler**
- ♦ The Modeling Perspective
- ♦ Views in the Modeling Perspective
- ♦ Rational Software Modeler Projects



3

This section introduces the product, and the key features and roles in systems development.

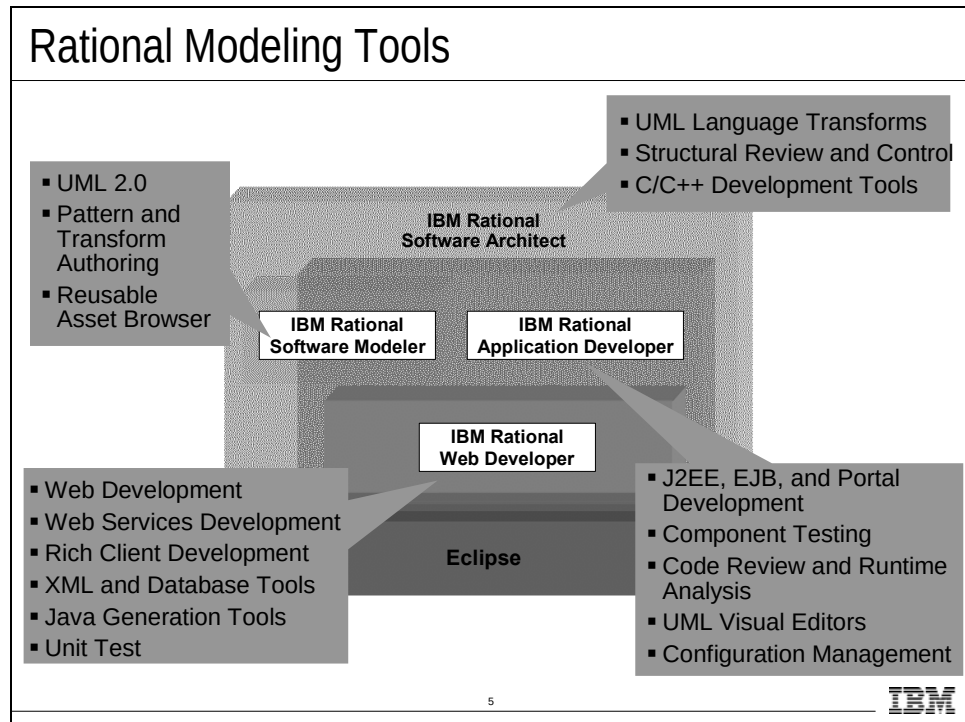
Comprehensive Solutions



The IBM Software Development Platform is based on an architectural foundation called the Eclipse universal framework. This framework enables unprecedented tool integration and artifact traceability throughout the development lifecycle.

What is Eclipse? It is both an open source software development project with over 100 active vendor participants and a rich foundation for providing shared services across an application tooling environment.

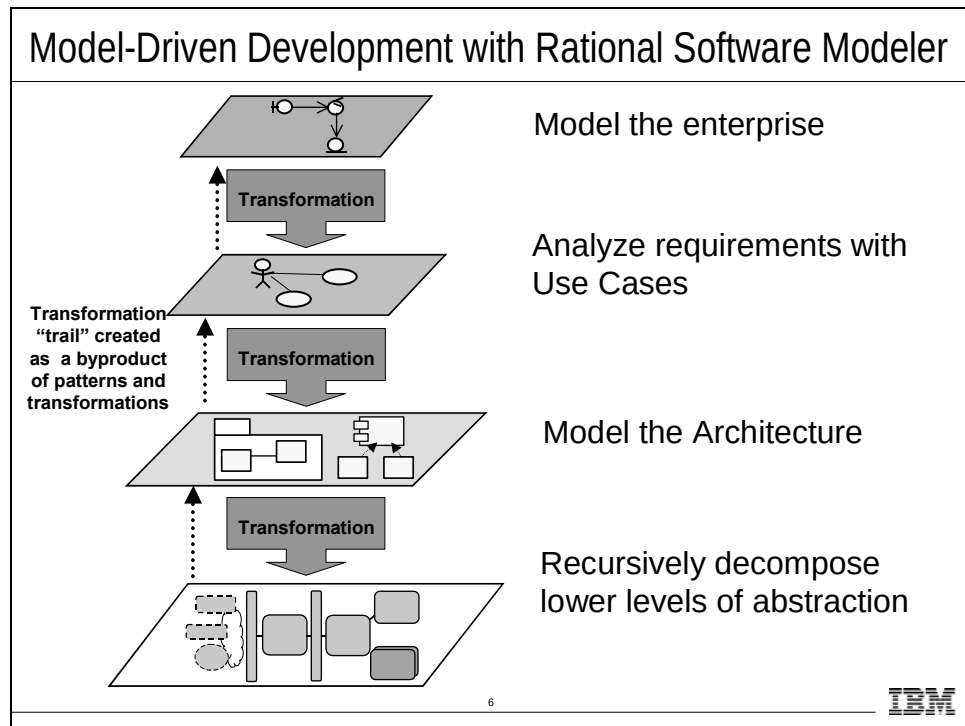
Rational Modeling Tools



Rational brand promotes an integrated and automated systems development experience across the team. In addition to offering world class functionality, each of these solutions shares some common benefits. Each of these solutions is:

- **Flexible:** You can match the tools to diverse skill sets in enterprise development efforts with no compromise required.
- **Modular:** You can have your tools precisely fit the needs of your organization.
- **Open:** Each represents the implementation of open industry standards, so you are not limited to interoperating with any set of proprietary standards.
- **WebSphere optimized:** Rational Application Developer and Rational Web Developer are optimized for WebSphere software and also provide capabilities for development on other technology platforms.

Model-Driven Development with Rational Software Modeler



Rational Software Modeler is designed to support model-driven development, which is the development of the appropriate models to facilitate all development activities and stages in the system development lifecycle, including tools to transform models into more detailed or more abstract models to move development work forward.

The systems engineer then uses Software Modeler to perform architectural analysis and delineates the logical subsystems of the system and the initial deployment or locality model.

The software architect can use Software Modeler to create a platform-independent design model from the use-case model. This model or set of models can be transformed in platform-dependent implementation models (including code and UML) with the assistance of visual development tools, such as:

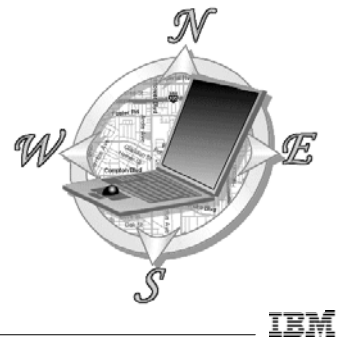
- UML editors for Java, C++, or data
- Site Designer
- Page Designers

As each new stage of system development begins, transformations can be used to create more detailed models that are incrementally closer to the target platform and infrastructure. Transformations can be designed to include traceability so that you can query the target model using elements from the source model to find elements in the target model. This feature is currently built into the UML-to-Java transformation that comes with Software Modeler, and can be included in any transformations that are custom-built. After the transformation is complete, you can right-click a model element in the design model and perform a query to find the associated Java code.

The Modeling Perspective

Where Are We?

- ♦ Introduction to Rational Software Modeler
- ♦ **The Modeling Perspective**
- ♦ Views in the Modeling Perspective
- ♦ Rational Software Modeler Projects

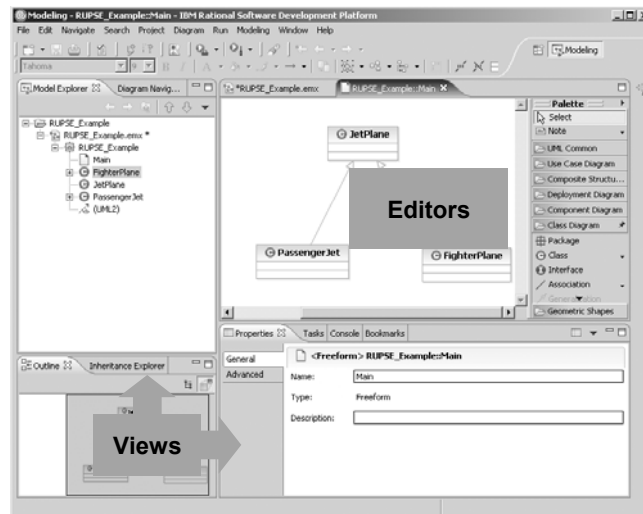


This section discusses the **Modeling** Perspective in Rational Software Modeler and the views it contains.

The Modeling Perspective

The Modeling Perspective

Perspective: Defines the initial set of content editors and related views and their layout.



The workbench window contains one or more perspectives. A **perspective** defines the initial set of content editors and related views (used to navigate information in the editor) and their layout in the workbench window.

Perspectives gather together sets of features for specific development activities. For example, the Java perspective gathers tools for writing code—the Package Explorer, Problems view, Outline view, and Java editor.

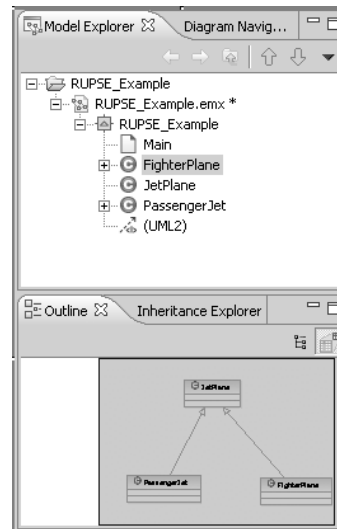
The Modeling perspective contains tools required for building UML models and diagrams, including the diagram editor, with the following views: Model Explorer view, Pattern Explorer, Outline view, Inheritance Explorer, Properties view, Tasks view, Output window, and Bookmarks view.

Views

Views

Views exist in the **Modeling Perspective**:

- For exploring modeling projects:
 - Model Explorer view
 - Diagram Navigator
- For exploring resources:
 - Pattern Explorer view
 - Outline view
 - Inheritance Explorer view
 - Properties view
- For getting feedback:
 - Tasks view
 - Output window



9



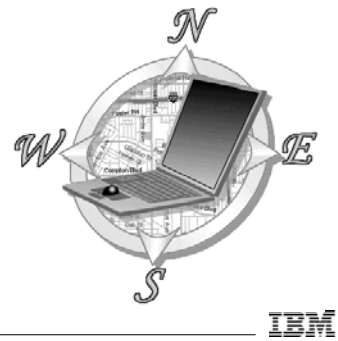
Views are windows that provide different ways to navigate the information in the workspace and in the workbench. For example, the Model Explorer view displays projects and models you are working with. A diagram in the Model Explorer view can be opened in the diagram editor. The properties of the selected diagram can be modified in the Properties view.

A view might appear by itself or stacked with other views in a tabbed notebook. You can change the layout of a perspective by opening and closing views and by docking them in different positions in the Workbench window.

Views in the Modeling Perspective

Where Are We?

- ◆ Introduction to Rational Software Modeler
- ◆ The Modeling Perspective
- ◆ **Views in the Modeling Perspective**
- ◆ Rational Software Modeler Projects



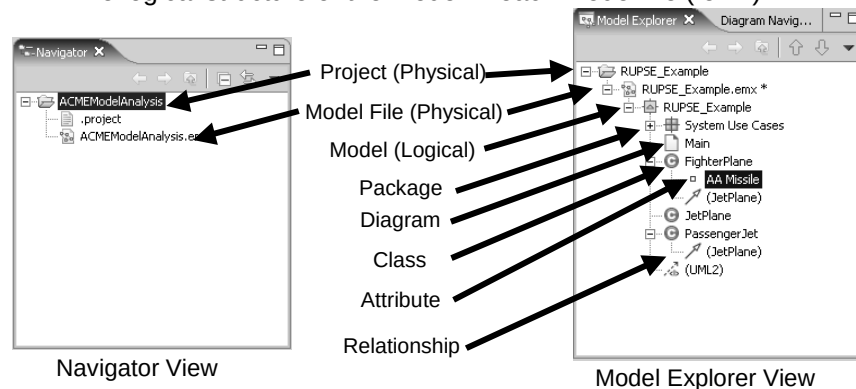
10

This section highlights the editors and views that are unique to the Modeling perspective.

Model Explorer View

Model Explorer View

- ♦ Used for navigating and organizing models.
- ♦ Shows:
 - Physical projects in the workspace and the model files they contain
 - The logical structure of the model in each model file (.emx)



11



The Model Explorer view shows all the model (.emx) files in the project, along with the projects and file structures they reside in.

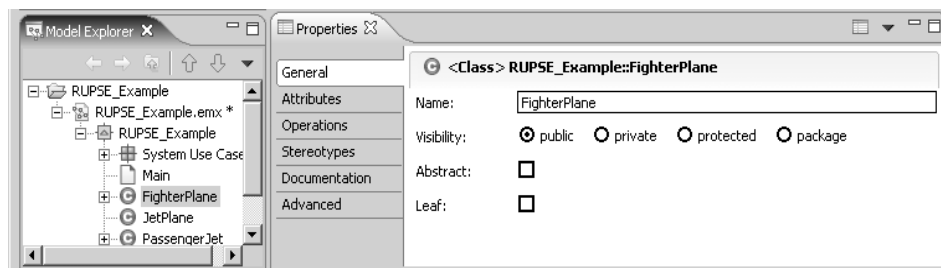
Under each model file, the Model Explorer view displays the logical structure of the model, including the name of the model as well as any packages, diagrams, relationships, and other model elements it contains. The model icon (light brown, containing a triangle) inside the model file allows you to add model elements at the root level of the model, and it allows you to set certain properties for the model.

You can use the Model Explorer view to add, delete, move, organize, and sort model elements for each model in your project.

Properties View

Properties View

- ◆ Displays properties and values for the selected element.
- ◆ Default properties view pages:
 - General
 - Documentation
 - Advanced



12

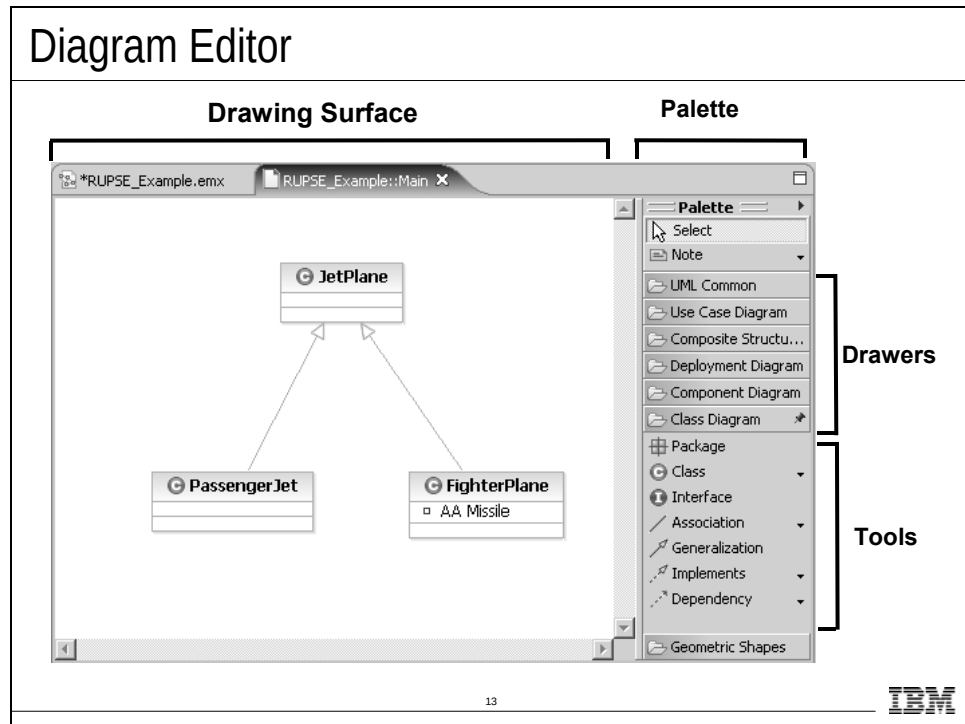


All models, diagrams, model elements, and relationships have associated properties, just like all other project resources.

In the **Modeling** Perspective, properties are organized into a series of tabbed categories. Most model elements contain the following categories:

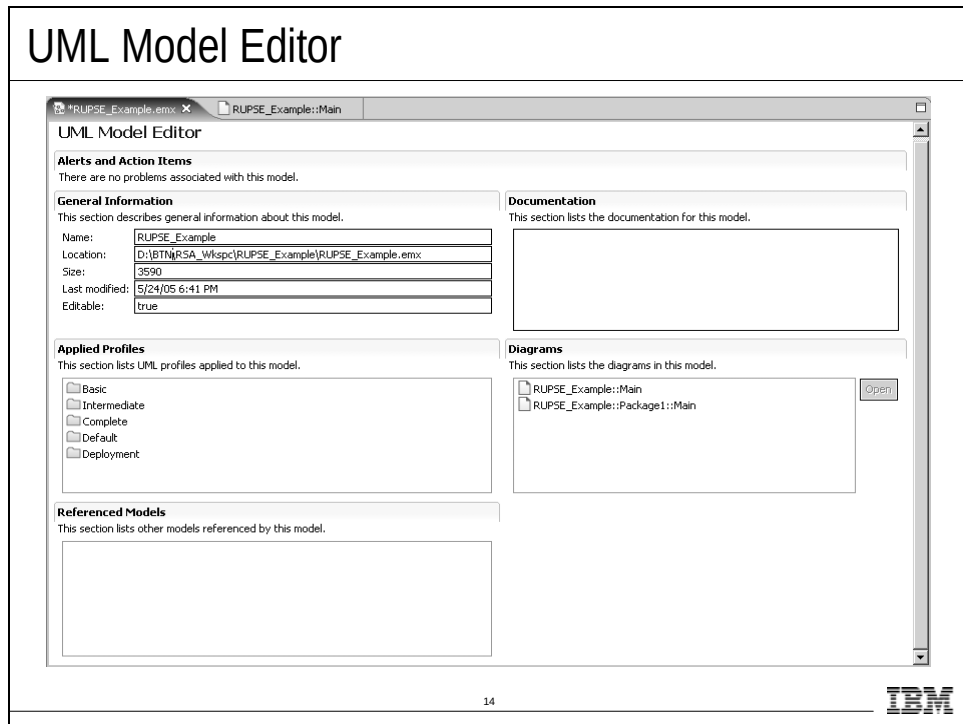
- **General:** Includes the name of the elements, qualified name, keywords for searches, visibility, and an option to make the element abstract.
- **Documentation:** Used for adding documentation to model elements.
- **Advanced:** Displays an Eclipse-style properties page.

Diagram Editor



The diagram editor contains both the drawing surface and the toolbox. The toolbox contains a basic set of tools (available by default) for selecting and annotating shapes in the diagram, along with one or more drawers of tools for various diagram types. You can customize the look and behavior of drawers in the toolbox, as well as create and customize your own drawers.

UML Model Editor



The UML Model editor appears every time you open a model. To close your model, close the UML Model editor. From the UML Model editor, you can add documentation to the model and navigate to specific diagrams in the model. It also provides the following model information:

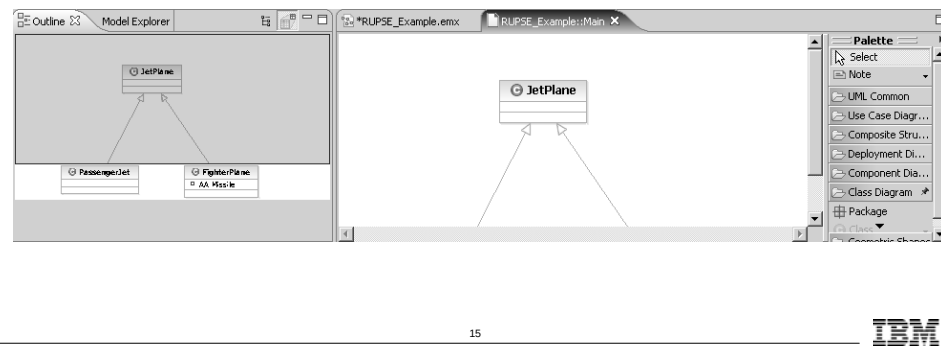
- **General Information:** Provides general model file properties, including name, location in the workspace, size of the file, modify date, and read or write status.
- **Applied Profiles:** Shows which UML profiles are currently applied to the model. Applying profiles makes available sets of stereotypes that extend the UML to create specialized models.
- **Referenced Models:** Shows which models the open model references. A referenced model contains model information from a referenced component or project.
- **Documentation:** Allows you to add documentation at the model level, just the way you would document an element in a model.
- **Diagrams:** Allows you to navigate directly to any diagram in the model and open it in the diagram editor.

Outline View

Outline View

When the diagram editor is open, the Outline view shows:

- A thumbnail view of the diagram.
- A list of elements displayed in the diagram.



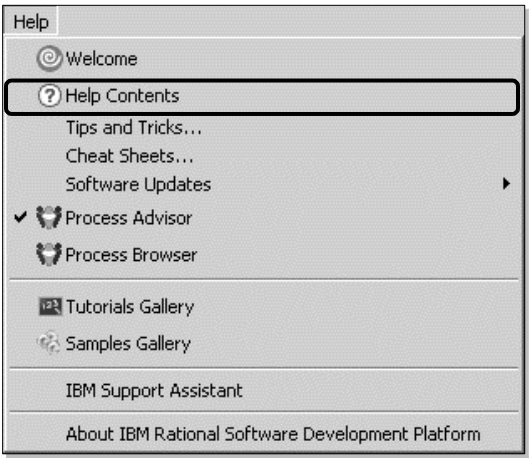
The Outline view displays an outline of a structured file that is currently open in the editor area and lists structural elements.

The contents of the Outline view are editor-specific. When the diagram editor is open, the Outline view can show either a tree view of the elements displayed in the diagram or provide a thumbnail of the diagram with the screen area shown in gray. You can move the active screen area in the outline view to navigate the diagram. The tree view displays an outline of a structured file that is currently open in the editor area and lists structural elements.

Help and Online Training Resources

Help and Online Training Resources

- ◆ Help menu includes:
 - Tips and Tricks
 - Cheat Sheets
 - Samples Galleries
- ◆ Help contents includes:
 - Overview
 - First Steps
 - Tutorials
 - Samples
 - Online Help



16



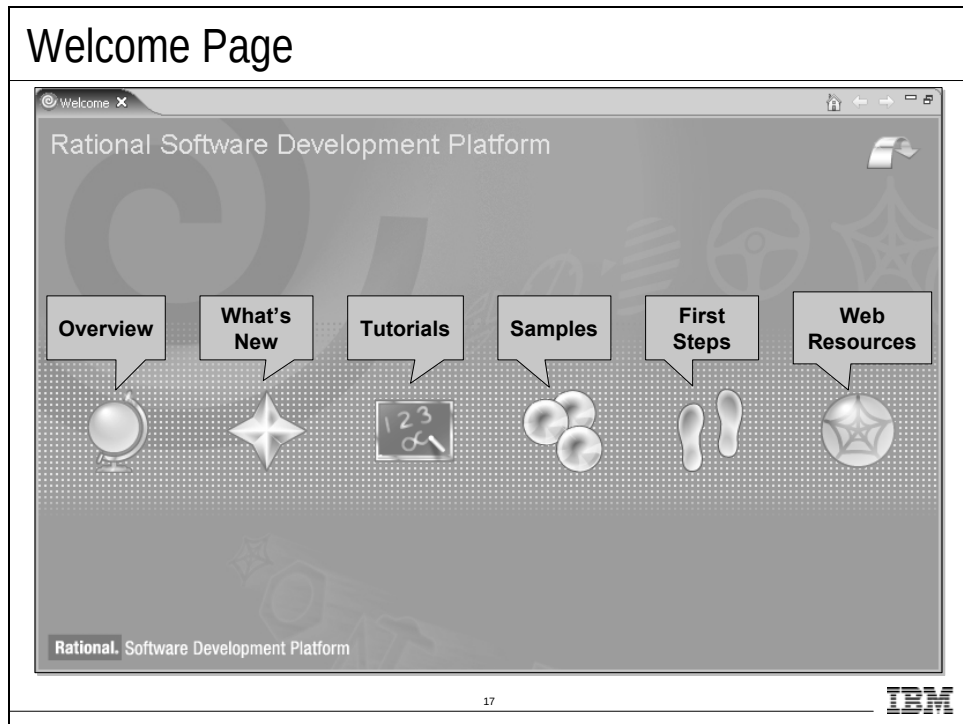
The help resources built into the Rational Software Modeler development environment include:

- **Tips and Tricks:** A list of tips for advanced users.
- **Cheat Sheets:** Step-by-step instructions for common tasks. Opens a Cheat Sheets view in the **Modeling** Perspective.
- **Samples Galleries:** A wide range of sample applications featuring different technologies.

From the help contents, you can find:

- **Overview:** Presents a series of animated tours, including presentations on modeling, developing code, best practices, and the environment.
- **First Steps:** Provides step-by-step guidance for building an application.
- **Tutorials:** Provides in-depth instructions on various topics in three modes:
 - **Watch and Learn:** Provides feature demonstrations.
 - **Play and Learn:** Provides interactive feature simulations.
 - **Do and Learn:** Provides hands-on instructions for learning to do various tasks.
- **Samples:** Provides a gallery of UML models, patterns, scripts, and transforms you can use as the basis for your own projects and resources.
- **Online Help:** Contains a “tips and tricks” collection and a collection of reference “cheat sheets.”

Welcome Page



The Welcome page offers the following resources on the Rational Software Development Platform (supporting Rational Software Modeler and Rational Application Developer for WebSphere):

- **Overview:** A video tour of the development platform and descriptions of how to use the platform for different types of development.
- **What's New:** Descriptions of what new features have been added in the current version.
- **Tutorials:** A large gallery of tutorials of three types: "Watch and Learn," "Play and Learn," and "Do and Learn."
- **Samples:** Three types of code samples: simple application samples, applications based on specific technologies, and large and extensive showcase samples.
- **First Steps:** Instructions for getting started with various project types.
- **Web Resources:** A wide variety of IBM Rational brand Web resources.

Rational Unified Process Resources

Rational Unified Process Resources

Process guidance is built in to Rational Modeling Tools:

- Process Advisor
- Process Browser

The screenshot displays the IBM Rational Software Modeler interface. The main workspace shows a UML class diagram with three classes: **JetPlane**, **PassengerJet**, and **FighterPlane**. **PassengerJet** and **FighterPlane** are subclasses of **JetPlane**, indicated by solid line arrows with hollow triangle heads. **FighterPlane** has an association with **AA Missile**. The right-hand side features a **Palette** with various UML diagram elements like **Select**, **Note**, **UML Common**, **Use Case Diagram**, **Composite Structu...**, **Deployment Diagram**, **Component Diagram**, **Class Diagram**, **Package**, **Class**, **Interface**, **Association**, **Generalization**, and **Geometric Shapes**. The bottom of the interface has a **Process Advisor** tab, which is active. It shows a search scope of **Tool Mentors, Artifacts, Activities** and a list of 11 tool mentors, including **Setting Up for a Project Using Rational Software Architect**, **Identifying Design Elements Using Rational Software Architect**, **Designing Subsystems Using Rational Software Architect**, **Structuring the Implementation Model Using Rational Software Architect**, **Performing Use-Case Analysis Using Rational Software Architect**, **Performing Architectural Analysis Using Rational Software Architect**, **Designing Use Cases Using Rational Software Architect**, **Designing Classes Using Rational Software Architect**, and **Structuring the Use-Case Model Using Rational Software Architect**. The IBM logo is in the bottom right corner.

The IBM Rational Unified Process®, or RUP®, is a software engineering process and an industry-wide platform for best practices.

There is a configuration of the RUP designed especially for Software Modeler users, which is accessible in two ways:

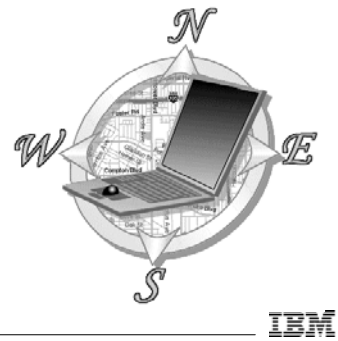
Process Advisor: The Process Advisor is a view that presents context-sensitive references to Rational Unified Process Tool Mentors, Artifacts, and Activities. From this view, you can also search the RUP and access Help.

Process Browser: The RUP Process Browser is a separate window that opens to display the RUP content selected in the Process Advisor view and also allows you to browse the RUP.

Rational Software Modeler Projects

Where Are We?

- ♦ Introduction to Rational Software Modeler
- ♦ The Modeling Perspective
- ♦ Views in the Modeling Perspective
- ♦ **Rational Software Modeler Projects**



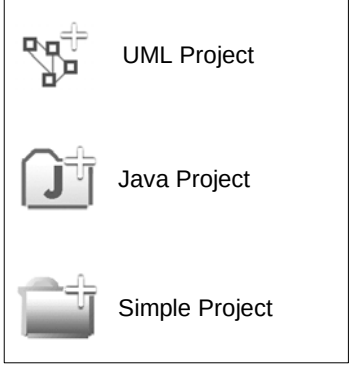
19

This section introduces the available project types and model templates.

Rational Modeling Tools Project Types

Rational Modeling Tools Project Types

- ♦ Two basic types:
 - UML projects:
 - For pre-implementation models
 - Blank or based on a model template
 - Implementation projects:
 - Java, EJB, and so on
 - Implementation projects can contain UML class and sequence diagrams
- ♦ Simple projects can be of either type.



New Project Wizard

20


A project in Eclipse and Rational Software Architect, Rational Software Modeler, and Rational Systems Developer is just a container for resources, such as model files, code files, HTML or XML files, and so on.

In Software Modeler, you will develop two types of projects, **UML projects** for pre-implementation models and **implementation projects**, in which the code is the implementation model. Implementation projects can contain free-standing UML class and sequence diagrams, used to visualize the code's structure and runtime behavior.

A **Simple** project is an empty project folder, which you can populate with any file structures and resources.

A **UML project** is a Simple project that contains a model file. Models can be created either blank or based on a Software Modeler UML model template.

The following implementation projects are available:

Enterprise: An enterprise application project contains the hierarchy of resources required to deploy a J2EE enterprise application, often referred to as an EAR file.

EJB: EJB projects contain the resources contained in an EJB module.

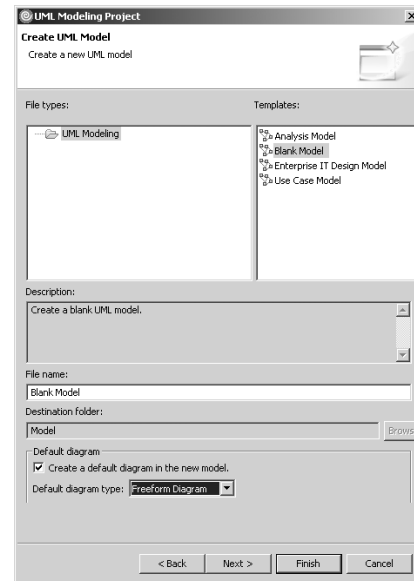
Web: Web projects are used for creating dynamic Web applications and static Web sites. Dynamic Web projects provide an accompanying EAR project.

Connector: A connector is a J2EE standard extension mechanism for containers to provide connectivity to enterprise information systems (EISs).

Creating a New Model

Creating a New Model

- ◆ UML Project templates are available in the New Project wizard.
- ◆ The following model templates are available:
 - Analysis Model
 - Blank Model
 - Enterprise IT Design Model
 - Use-case Model



21



In Software Modeler, model files are used to create **“pre-Implementation”** models. They contain conceptual UML content that does not *directly* reflect implementation artifacts. They present just enough detail so that the model can be easily understood. Model files contain both the UML semantics of the model and UML diagrams that depict the semantic elements.

When creating a UML Modeling project, a page in the New Project Wizard offers the opportunity to select a UML model template or blank model for the new project. After the project has been created, you can add additional model resources or delete the original model resource if necessary.

Demo: Creating a Simple Model

Demo: Creating a Simple Model

- ♦ The instructor will now show you how to:
 - Create a UML project.
 - Create a Blank model.
 - Create a Class diagram.
 - Validate the diagram.



22

IBM

Watch your instructor create a simple UML model.

Lab: Create a UML Model

Lab: Create a UML Model

Tasks:

- Create a new UML project.
- Create a Blank model.
- Create a Class diagram.
- Validate the diagram.



23

IBM

Complete Lab 1: Getting Started.

Review

Review

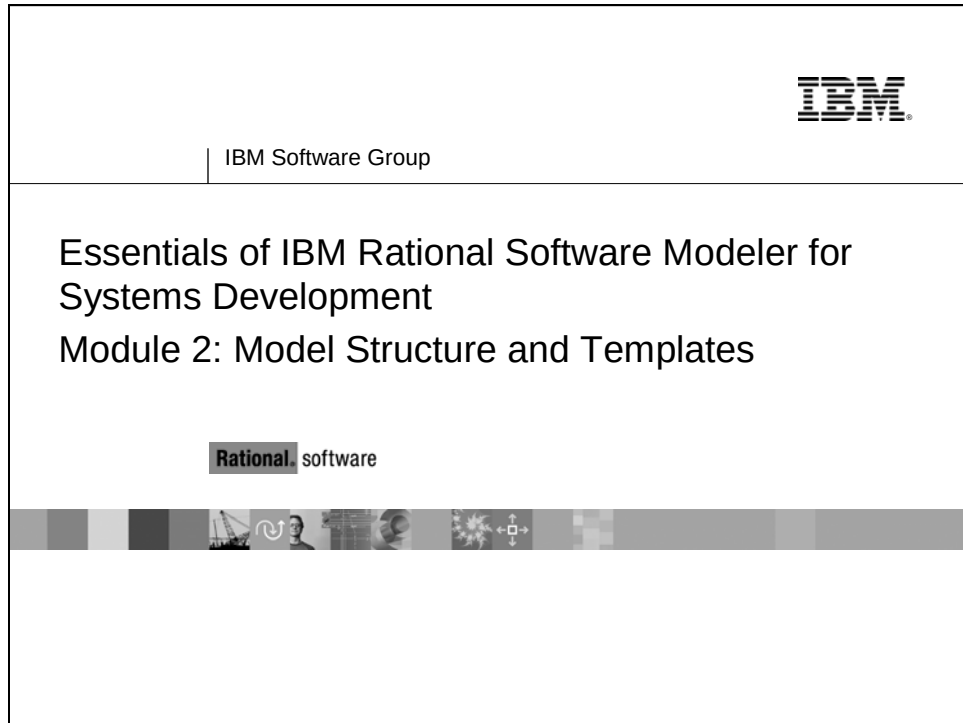
- ♦ What is a “view”?
- ♦ What is a “perspective”?
- ♦ What project type is used for “pre-implementation” models?



24

IBM

► ► ► **Module 2**
Model Structure and Templates



Topics

Model Templates in Rational Software Modeler	2-3
Demo and Lab	2-9

Objectives

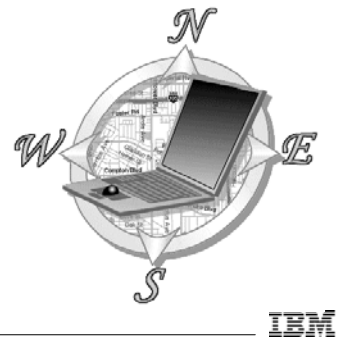
Objectives

- ◆ After completing this module, students will be able to:
 - Use template models and model building blocks provided by Rational Software Modeler to reduce repetitive labor.
 - Build and organize models with Rational Software Modeler to display a system structure.

Model Templates in Rational Software Modeler

Where Are We?

- ♦ Model Templates in Rational Software Modeler
- ♦ Demo and Lab

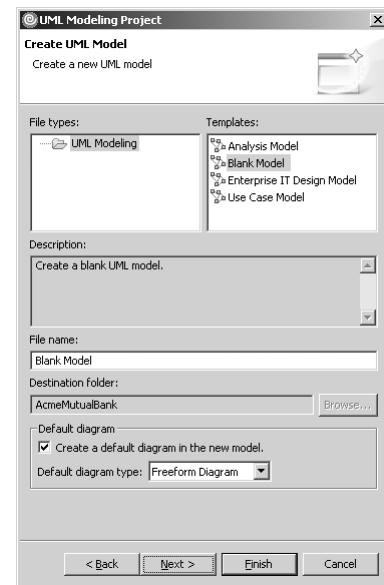


3

Model Templates in Rational Software Modeler

Model Templates in Rational Software Modeler

- ◆ Templates provide a building block for starting.
- ◆ Templates include:
 - Analysis Model
 - Blank Model
 - General design models
 - Freeform modeling
 - Enterprise IT Design Model
 - Design model specifically for n-tier business applications
 - Use-case Model



4



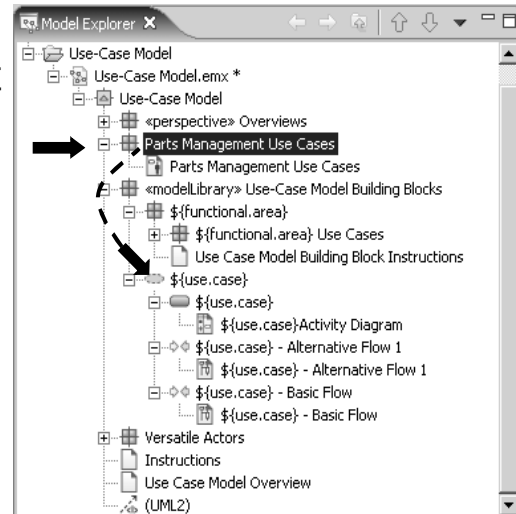
Rational Software Modeler provides the following model templates:

- **Analysis Model:** Contributes the Software Engineering Process profile, and some default content, such as packages and diagrams.
- **Blank model:** Includes no default content other than a Main freeform diagram and no profiles applied. The blank model template allows you to create your own model types and structures.
- **Enterprise IT design model (EITDM):** Contributes default content, including a profile that supports the modeling of services and transformations of designs into code. Use this template for design (and optionally for analysis) when targeting business applications. To create a design model for a simpler application, start with a blank model instead and add the appropriate package structure.
- **Use-case model:** Includes blank model with default content (use cases, package structure, diagrams) for building a use-case model.
- In the design phase, you can also use the blank template to construct an **Implementation Overview** model to capture the high-level organization of the implementation or planned dependencies among implementation projects and packages. An Implementation Overview model in Rational Software Modeler corresponds with the Implementation model in the RUP.

Creating a Model Using a Model Template

Creating a Model Using a Model Template

- ♦ Create a UML project and select a template.
- ♦ Populate the model with:
 - UML packages
 - UML elements based on model building blocks provided



5



Every template contains a «modelLibrary» package called *TemplateName Building Blocks*. This package contains pieces of model content that you can use to build the design model more quickly. Building blocks act as “template” model elements. You can copy and paste the building block elements to create new elements for your model.

To use a building block element:

- In the Model Explorer, copy and paste or CTRL-drag a building block element from the building blocks package to the desired location in the model.
- Right-click the new element and choose Find/Replace to change the placeholder name (\$name) to the desired name.

A best practice for naming diagrams is to create a descriptive name and then add the diagram type. For example, the use-case diagram in the slide is called “Parts Management Use Cases.” It might also be called “Parts Management Use-Case Diagram.”

Model Templates: Use-Case Model

Model Templates: Use-Case Model

A Use-Case Model is a model of the system's intended functions and its environment.

Use Case

Unit of externally visible functionality

Activity

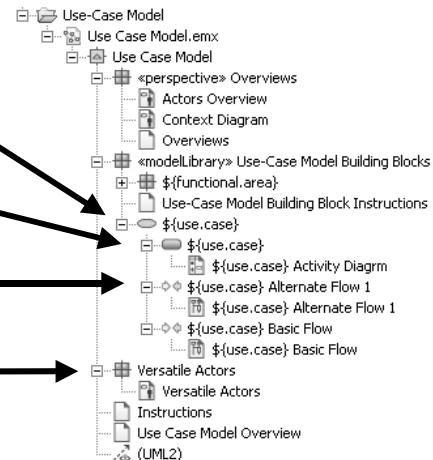
For gathering activity diagrams for the use case (optional)

Interactions

For modeling each flow of events in the use case using interaction diagrams

Versatile Actors

Actors that participate in multiple functional areas of the application



6



The Use-Case Model is a model of the system's intended functions and its environment. The Use-Case Model template in Rational Software Modeler contributes the default content shown in the slide. The model is divided with packages into the functional areas within the application. Each functional area package contains actors and use cases. Each use case can optionally contain interactions that link directly to use-case specifications and contain activity diagrams showing each flow of events graphically. The Use-Case Model is used as an essential input to activities in analysis, design, and test.

The model contains a «perspective» package containing overview diagrams of all the functional areas in the model and a package containing “versatile actors,” which are actors that appear in multiple functional areas in the application.

To create your Use-Case Model using the building blocks, you perform these activities iteratively:

- Create "functional area" packages using the package building block provided.
- Populate each package with the use cases for that functional grouping using the use-case building block provided.

You can then perform the other activities of use-case modeling, such as gathering use cases into use-case diagrams, adding actors, drawing extend or include relationships among the use cases, drawing associations from the use cases to the actors that participate in them, writing descriptions for the use cases and actors, and (where it adds value) composing the high-level activity and "black box" sequence diagrams for each use case.

For information on the Use-Case Model, see the Rational Unified Process: **Artifacts > Requirements Artifact Set > Use-Case Model**.

Model Templates: Analysis Model

Model Templates: Analysis Model

An Analysis Model describes the realization of use cases to analysis classes.

- Provides an abstraction of the design model

Boundary Class

Mediates between the system and something outside the system

Controller Class

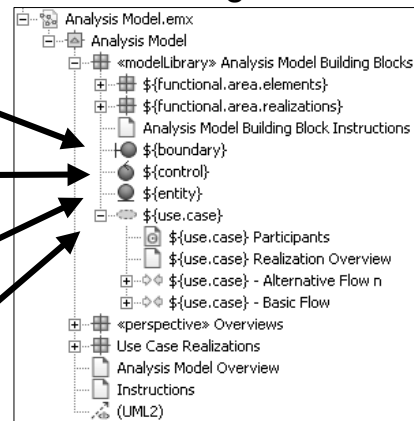
Provides the coordinating behavior in the system

Entity Class

Reused in many use cases, often with persistent characteristics

Use-Case Realization

Shows how use cases are implemented in terms of collaborating objects



7

IBM

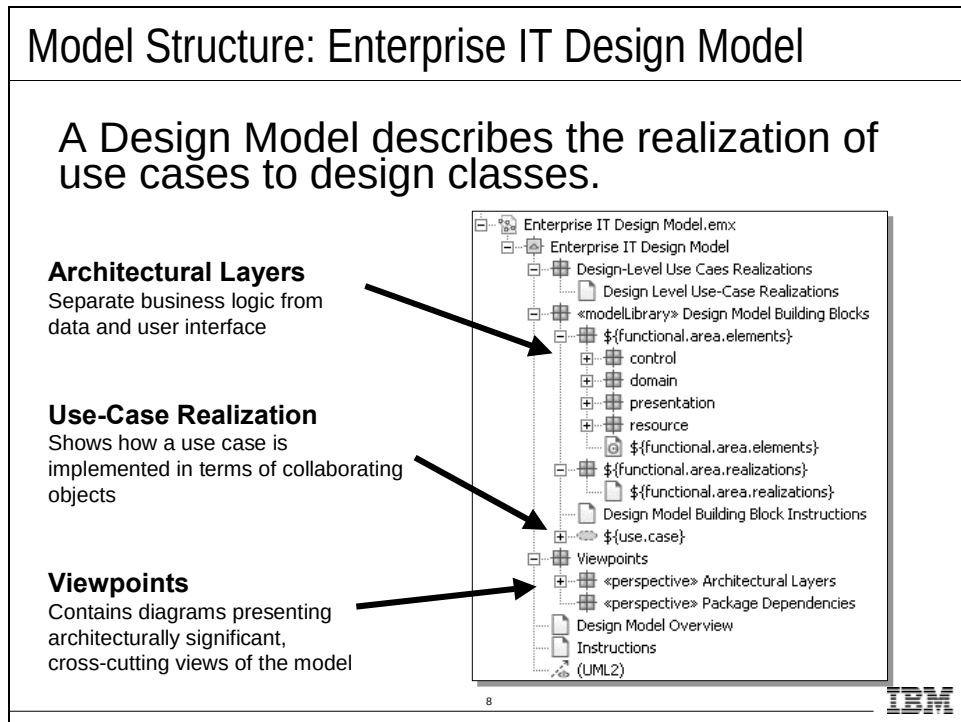
The Analysis Model is an object model that describes the realization of use cases (shows the collaborating objects derived from flows of events modeled in the use-case model). The Analysis Model contains the results of use-case analysis. The Analysis Model serves as an abstraction of the Design Model.

In Rational Software Modeler, the Analysis Model template contributes the default content shown in the slide. An “Analysis” profile is applied to model files created from this template.

The Analysis Model includes packages for the same functional areas defined in the Use-case Model, containing analysis classes and diagrams, derived from use-case realizations.

The process of use-case realization is beyond the scope of this course, though it is described in the Rational Unified Process (located in **Designer role > Use-Case Design > Guidelines > Use-Case Realization**). For a review of the analysis model and analysis classes, see **Artifacts > Analysis & Design Artifact Set > Analysis Model**.

Model Structure: Enterprise IT Design Model



The Design Model is an object model describing the realization of use cases and serves as an abstraction of the Implementation Model and its source code. The Design Model is used as essential input to activities in implementation and test.

The “Enterprise IT Design Model” (EITDM) template contributes default content, including a profile that supports modeling of services and transformations of Design Model elements into code. This is the appropriate model type to use for design (and optionally for analysis) when targeting business applications and using code-generating transformations to support creation of such applications.

To create your Design Model using the building blocks, you perform these activities iteratively:

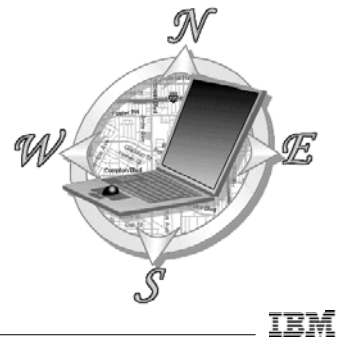
- Create functional area element and functional area realization packages.
- Populate the realization packages with specification elements as you specify the behaviors (typically interactions) that express the design-level realizations of your use cases. The "realization" packages define the usage contracts for your components and services.
- Populate the element packages with implementation elements that realize the specification elements in the realization packages.

For more information on the design model, see the Rational Unified Process, **Artifacts > Analysis & Design Artifact Set > Design Model**.

Demo and Lab

Where Are We?

- ♦ Model Templates in Rational Software Modeler
- ♦ Demo and Lab



9

Demo: Model Structure and Building Blocks

Demo: Model Structure and Building Blocks

- ♦ The instructor will now show you how to:
 - Create a basic model structure from a blank model.
 - Use a model building block.
 - Create a model building block.



10

IBM

Watch your instructor create a basic model structure.

Lab: Model Structure and Building Blocks

Lab: Model Structure and Building Blocks

- ◆ Use Rational Software Modeler to:
 - Build a model structure.
 - Use a model building block.
 - Create a model building block.



11

IBM

Complete Lab 2: Model Structure and Building Blocks.

Review

Review

- ♦ What model templates are available with Rational Software Modeler?
- ♦ How do you use a building block to minimize repetitive work?



12

IBM

Further Information

Further Information

- ♦ “Model Structure Guidelines for Rational Software Modeler and Rational Software Architect”
- ♦ RUP for Systems Engineering Plug-Ins, version 2.0

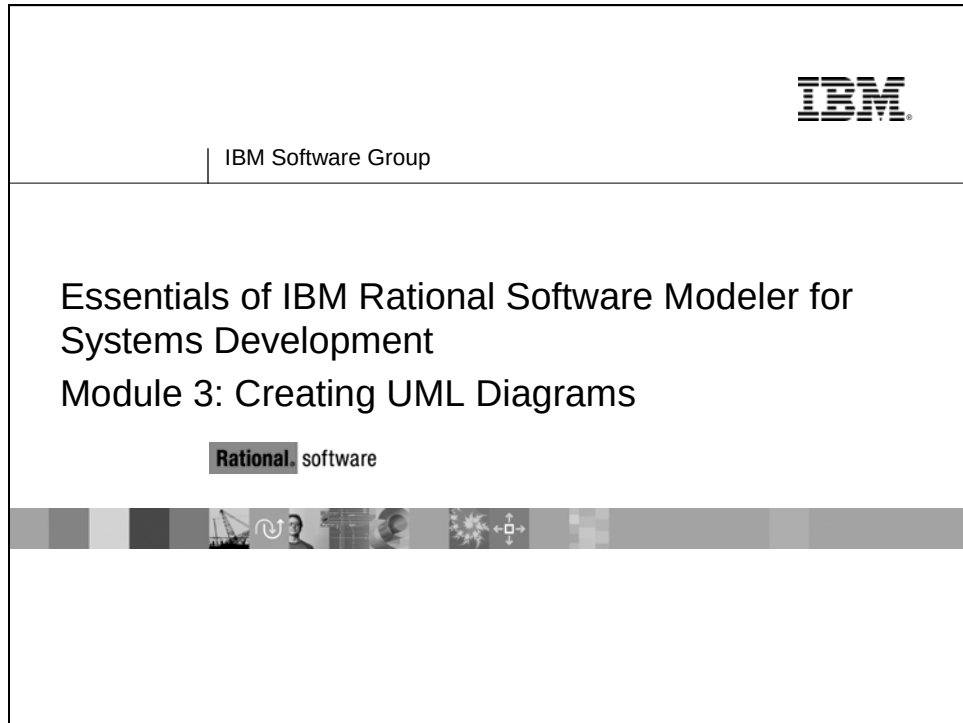
13



- IBM Rational Software: “Model Structure Guidelines for Rational Software Modeler and Rational Software Architect” (2004 Release).
- The purpose for the SE plug-in is to extend RUP to support systems development projects. Besides pure systems development projects, this plug-in is applicable to projects that have concurrent software and hardware development requiring more than one development team. It also provides content for handling architecturally significant deployment issues.

<http://www-128.ibm.com/developerworks/rational/library/4313.html>

► ► ► **Module 3**
Creating UML Diagrams



Topics

UML Diagrams.....	3-3
Drawing UML Diagrams	3-18

Creating UML Diagrams

Creating UML Diagrams

Objectives:

- Identify the UML diagrams available in Rational Software Modeler
- Describe Topic and Browse diagrams
- Create Use Case, Activity, Sequence, and Class diagrams in Rational Software Modeler

2



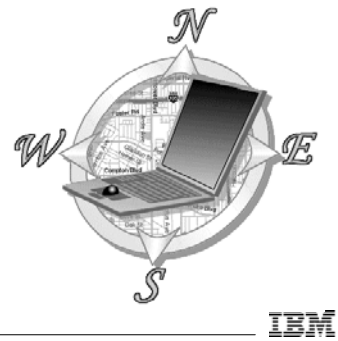
This module provides an overview of the UML diagram types available in Rational Software Modeler, with examples of the most commonly used diagrams, and explains how to use the diagram editor.

In the lab at the end of this module, you will add model elements and diagrams to an existing Design Model.

UML Diagrams

Where Are We?

- ♦ **UML Diagrams**
- ♦ Drawing UML Diagrams



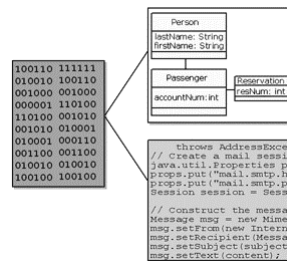
3

This section introduces UML diagrams.

What is a Model?

What is a Model?

- ♦ A model is a semantically closed abstraction of a subject system.
 - A model is defined in RUP as “a complete description of a system from a particular perspective.”
- ♦ Examples of models:
 - UML model
 - Code
 - Data model
 - Circuit Schematic
 - Mechanical Drawing



4

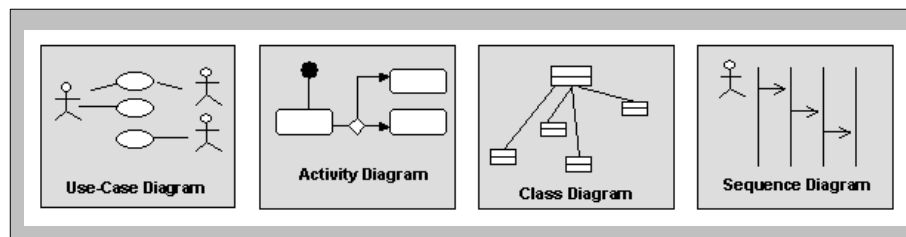


In the Rational Unified Process (RUP), a model is defined as a complete specification of a problem or solution domain from a particular perspective. Each model is *complete* in the sense that you do not need any additional information to understand the system from that perspective. A problem domain or a system can be specified by a number of models that represent different perspectives for different project stakeholders.

Diagrams

Diagrams

- ♦ Diagrams graphically depict a view of a part of your model.
- ♦ Different diagrams represent different views of the system that you are developing.
- ♦ A model element will appear on one or more diagrams.

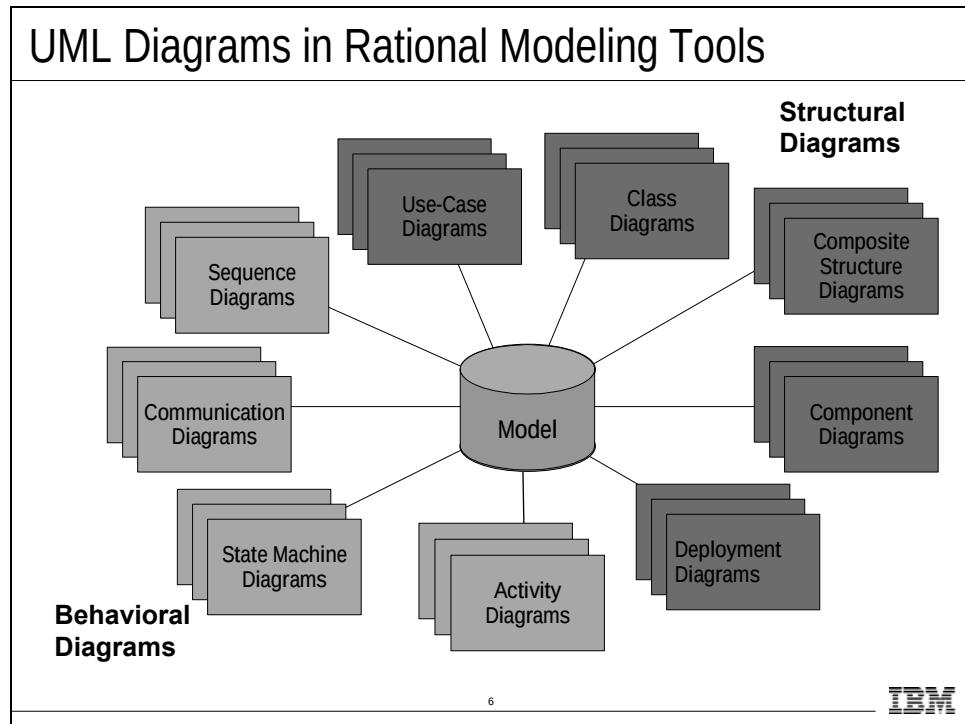


5

IBM

Diagrams provide a means of visualizing and manipulating the model elements.

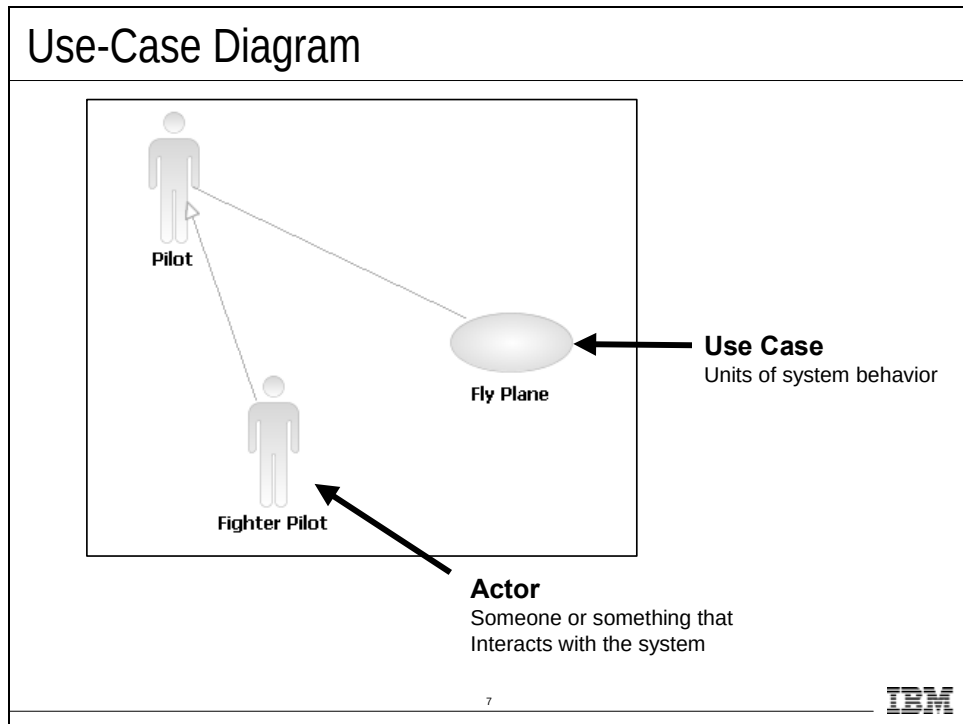
UML Diagrams in Rational Modeling Tools



Visual models of a system require many different diagrams to represent different views of the system for different project stakeholders. The UML provides a rich notation for visualizing models, including the following key diagrams:

- **Use-Case diagrams** to illustrate user interactions with the system.
- **Class diagrams** to illustrate logical structure.
- **Composite Structure diagrams** to show the internal structure of a class or component at runtime.
- **Component diagrams** to illustrate the organization and dependencies among modular parts of the system.
- **Deployment diagrams** to show the mapping of software to hardware configurations.
- **Activity diagrams** to illustrate flows of events.
- **State Machine diagrams** to illustrate the series of states an object can have.
- **Communication diagrams** to illustrate behavior in terms of how objects interact.
- **Sequence diagrams** to illustrate behavior in terms of the sequence of interactions between objects.

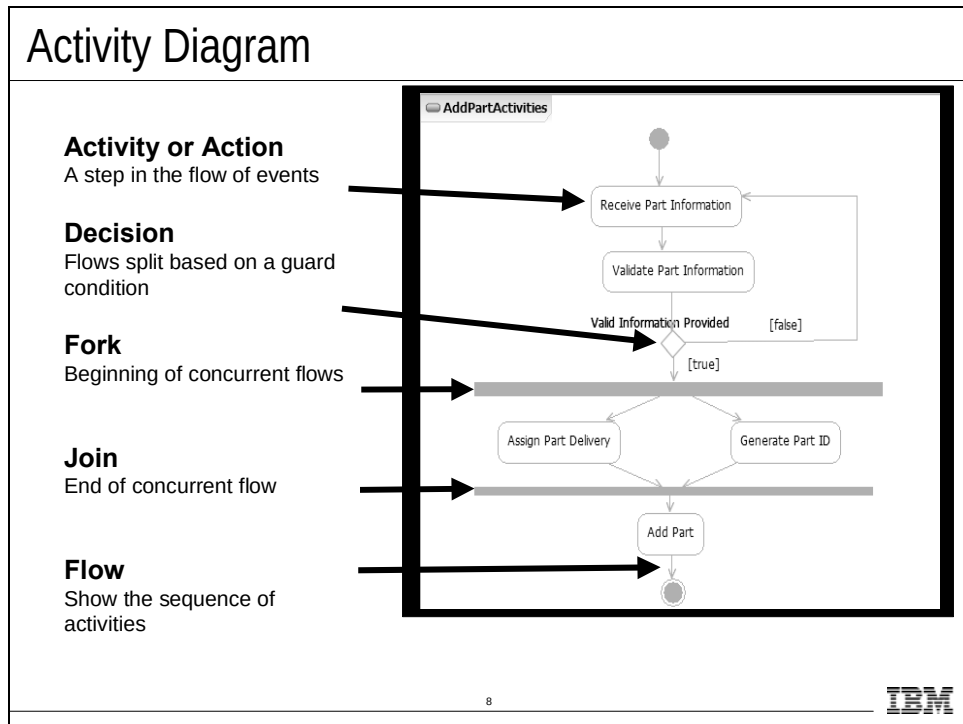
Use-Case Diagram



The use-case model contains the following elements, which are displayed in a use-case diagram:

- **Actors:** Used to represent someone or something outside the system that interacts with the system.
- **Use Case:** Used to represent a unit of system behavior that comprises several possible sequences of actions.

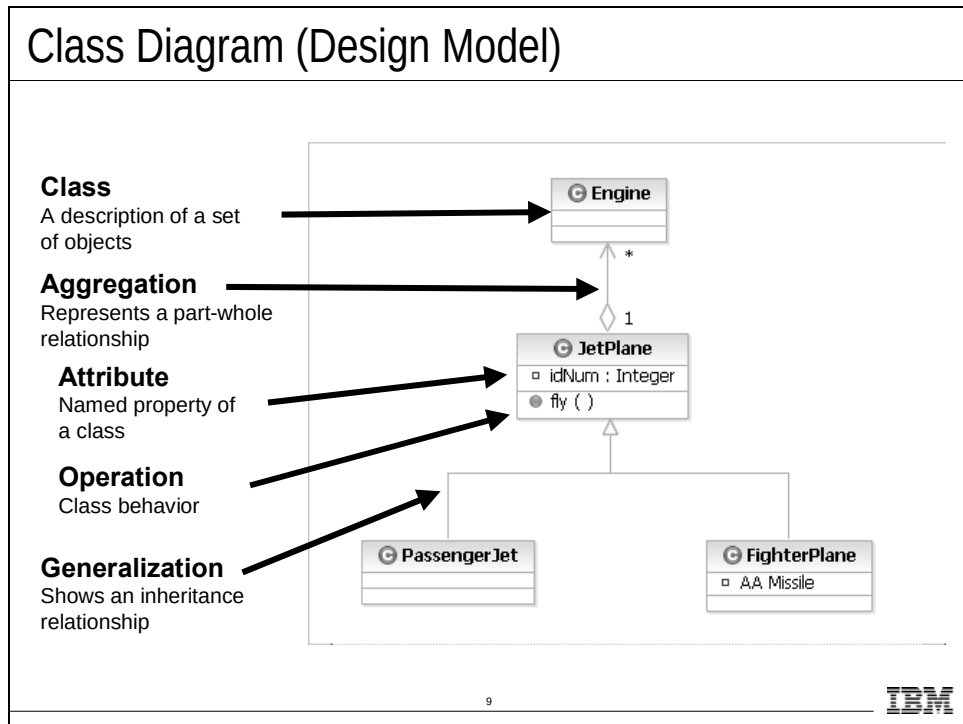
Activity Diagram



Activities graphically describe the flow of events of a use case. The flow of events consists of a sequence of activities that together produce something of value for the actor. The flow of events consists of a basic flow and one or several alternative flows.

- **Activity or Actions:** Represent the performance of an activity or action, a step within the flow of events.
- **Flow or Edge:** Show what activity state follows after another.
- **Decision or Merge** Control which flow (of a set of alternative flows) follows after the activity has been completed, based on a guard condition. Decisions are used to show alternative threads in the flow of events of a use case.
- **Forks or Joins:** Show the beginnings and ends of parallel subflows. Forks and joins are used to show concurrent threads in the flow of events of a use case.

Class Diagram (Design Model)

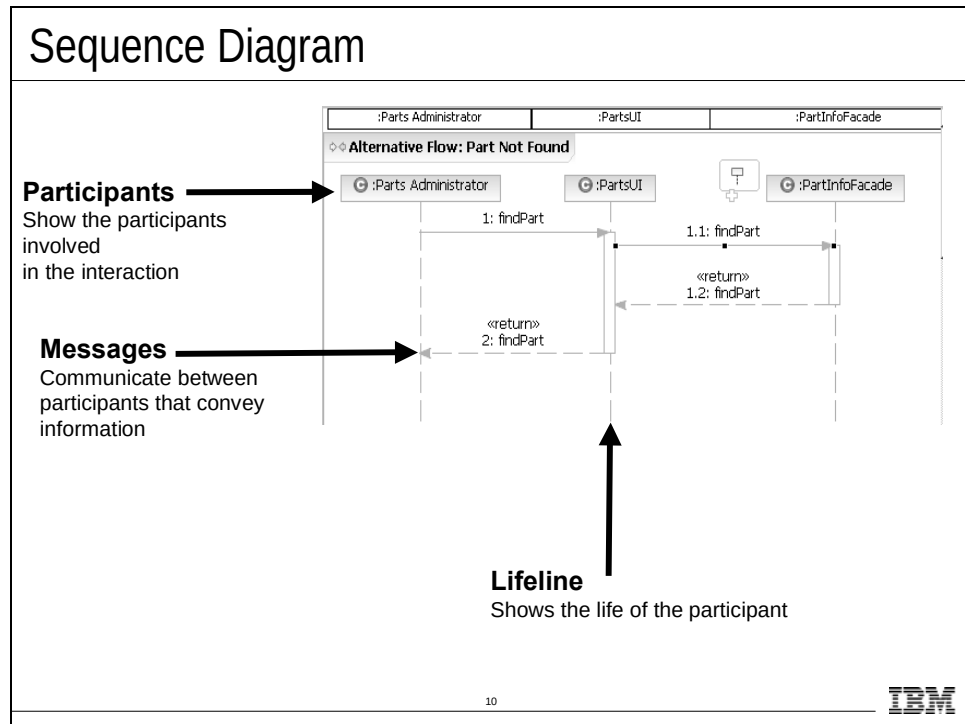


Class diagrams show the static structure of the model, in particular, its classes, their internal structure, and their relationships to other classes. Class diagrams do not show temporal information.

A class diagram typically displays the following elements and the relationships among them:

- **Classes:** Describe sets of objects that share the same responsibilities, relationships, operations, attributes, and semantics.
 - **Attributes:** Named properties of the class or its objects. An attribute also defines the type of its instances.
 - **Operations:** Services that can be requested (the behavior of the class).
- **Packages:** Can be included for the sake of context although UML 2.0 has brought package diagrams into the mainstream.

Sequence Diagram

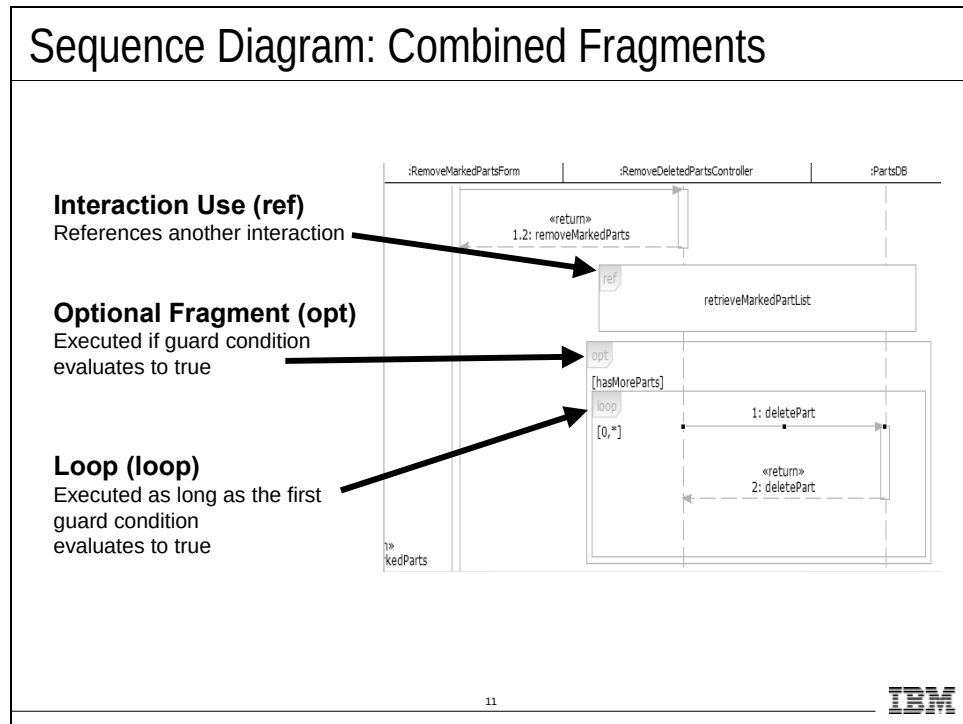


In the Design Model, sequence diagrams are used to show how participants interact to perform the behavior of all or part of a use case as part of a use-case realization. The sequence diagram clarifies the roles of participants in a flow of events, and thus it provides basic input for determining class responsibilities and interfaces.

Sequence diagrams display the following model elements:

- **Lifelines:** Represent an individual participant in the interaction.
- **Actors:** Can be included in the diagram when you want to show that an actor initiates the interaction.
- **Messages:** Communicate between participants that conveys information with the expectation that activity will ensue.

Sequence Diagram: Combined Fragments



To help with the modeling of, for example, multiple flows of events, which in the past always meant creating many interaction diagrams with only slight differences among them, you can use interaction fragment notation—new with UML 2.0 and supported in Rational Software Modeler—which allows you to embed multiple paths through an interaction inside the same interaction diagram. Interaction fragments have a keyword associated with them, shown on the frame surrounding the fragment.

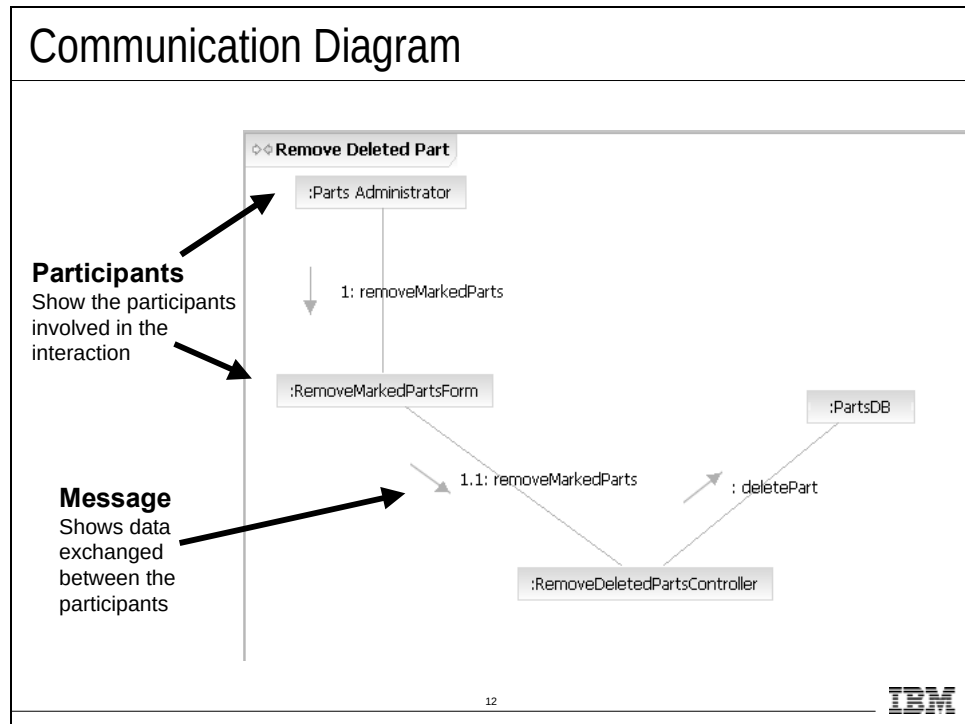
One type of interaction fragment, a combined fragment, in a sequence diagram has an operator and one or more subinteractions in a nested region of the diagram.

The more common combined fragment and interaction fragment types include:

- **loop:** The loop fragment contains one subfragment that executes as long as a Boolean guard condition evaluates to **true**.
- **ref:** The ref interaction fragment references another sequence diagram. In Software Modeler, adding the **ref** operator from the toolbox automatically generates a new interaction instance in the model.
- **opt:** The opt fragment type contains one subfragment that executes only when the guard condition evaluates to true.

See the Rational Software Modeler Help for discussion of other keywords.

Communication Diagram



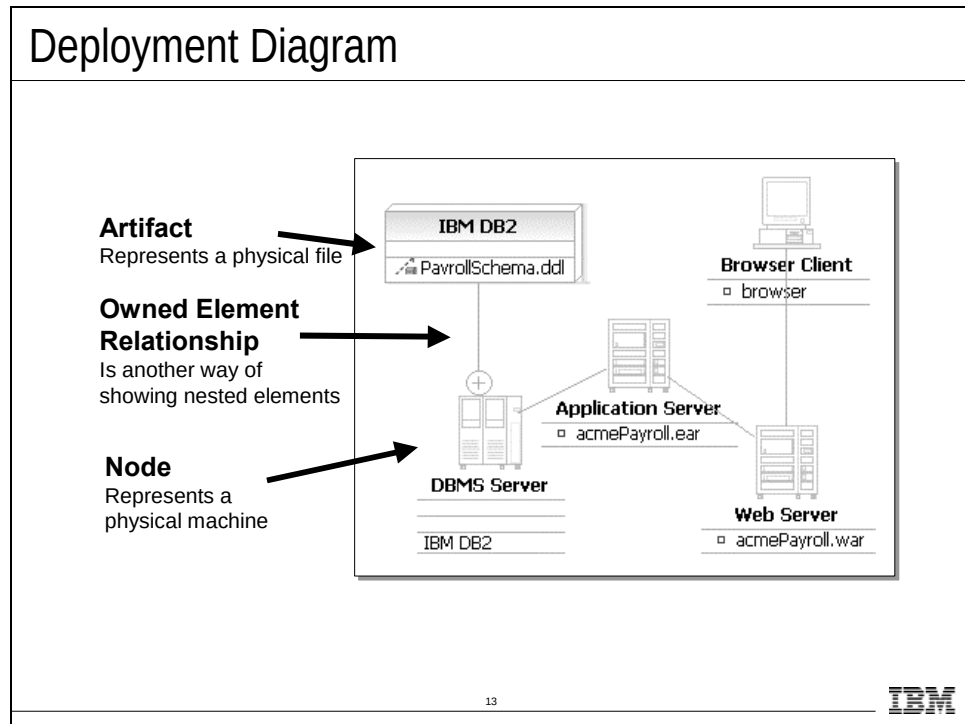
Communication diagrams provide another option to show how participants interact to perform the behavior of a particular use case or a part of a use case. Where sequence diagrams emphasize the interactions of participants over time, communication diagrams are designed to emphasize the relationships between participants.

- **Participants:** Participants show the name of the participant and its class, separated by a colon:
`participantname : classname`
- **Links:** Links are relationships among participants across which messages can be sent, shown as a solid line between two participants. A link can be an instance of an association, or it can be anonymous, meaning that its association is unspecified. Message flows are attached to links.

The diagram shown in the slide is based on the example sequence diagram from the previous slide. Two other elements that can appear in a communication diagram but do not appear in the example are:

- **Actors:** Actor instances show the invoker of the interaction.
- **Messages:** A message is a communication between participants that conveys information with the expectation that activity will ensue.

Deployment Diagram

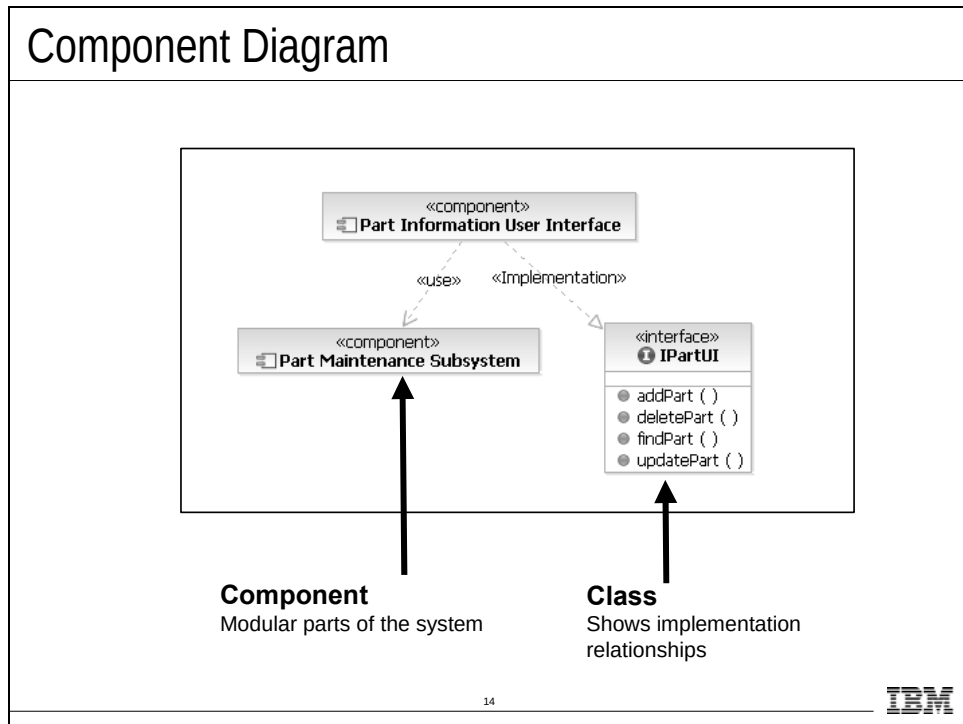


Deployment diagrams show the deployment architecture of the system, that is, which of the system's software artifacts reside on which pieces of hardware.

Deployment diagrams display the following model elements:

- **Artifacts:** Represent physical files (executable files, binaries, archives, and so on). In previous versions of UML, artifacts were called "components."
- **Nodes:** Represent any physical machine that can host software, such as a server or workstation. A node could also be used to model deployment to a virtual machine, for example, used as a test environment.

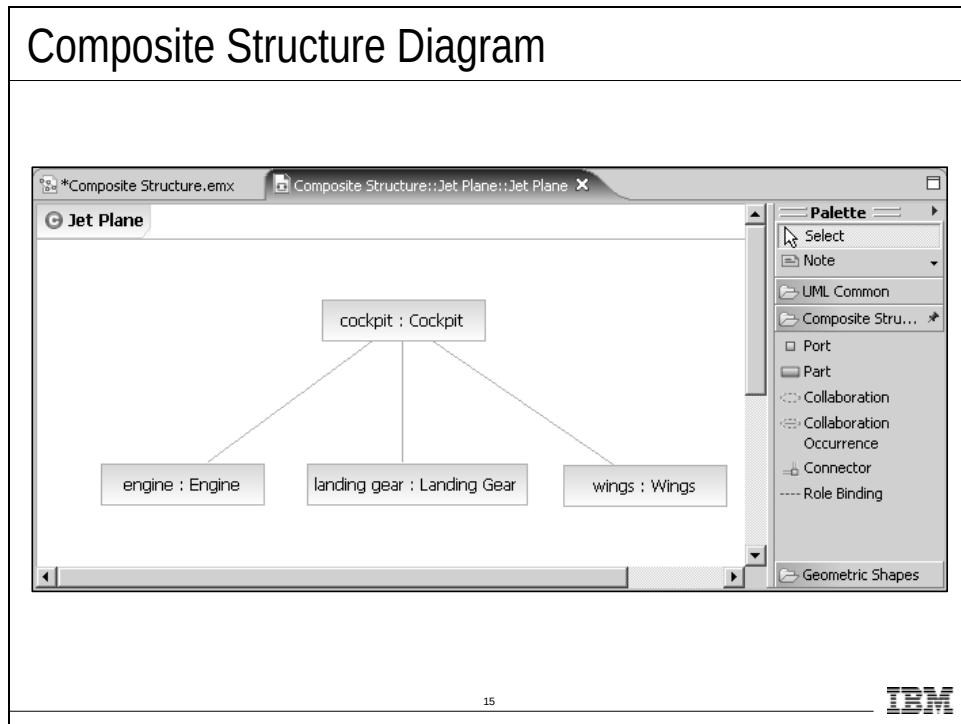
Component Diagram



Component diagrams show the runtime structure of the system at the level of software components. Components are the modular parts of the system and are made up of groups of related objects that are hidden behind an external interface.

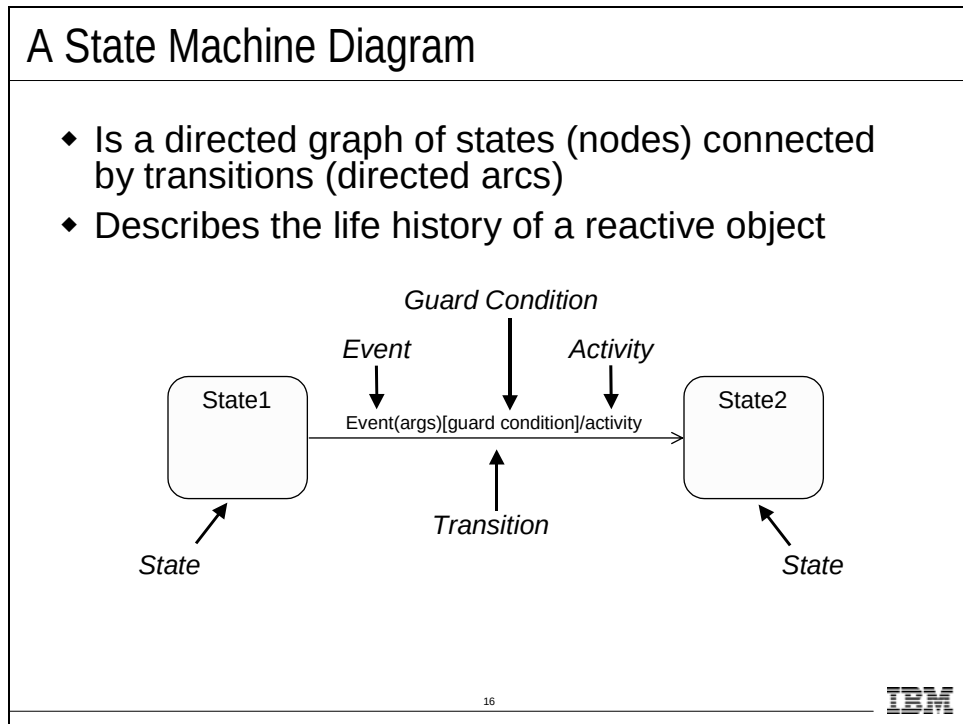
- **Component** notation is based on class notation, so a component can have attributes and operations.
- **Relationships** among components can include connectors or dependencies.

Composite Structure Diagram



As the system is further decomposed, each of the parts might be a structured class which contains parts themselves. This is a very effective method for visualizing the system architecture.

A State Machine Diagram



A state machine is a tool for describing:

- States the object can assume.
- Events that cause an object to transition from state to state.
- Significant activities that occur as a result.

A state machine is a diagram used to show the life history of a given class, the events that cause a transition from one state to another, and the activities that result from a state change. State machines emphasize the event-ordered behavior of a class instance.

The state space of a given class is the enumeration of all the possible states of an object.

A **state** is a condition in the life of an object. The state of an object determines its response to different events.

An **event** is a specific occurrence (in time and space) of a stimulus that can trigger a state transition.

A **transition** is a change from an originating state to a successor state as a result of some stimulus. The successor state could possibly be the originating state. A transition may take place in response to an event, and can be labeled with an event.

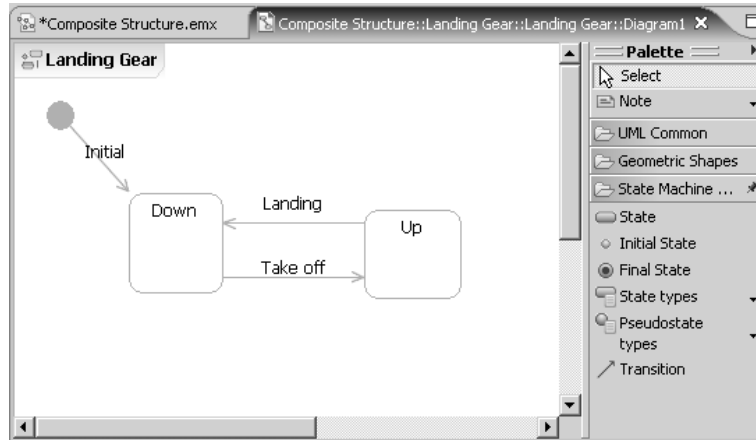
A **guard condition** is a Boolean expression of attribute values that allows a transition only if the condition is true.

An **activity** is a specification of executable behavior.

Example: State Machine

Example: State Machine

- ♦ Communicates the analysis and design of a project



17

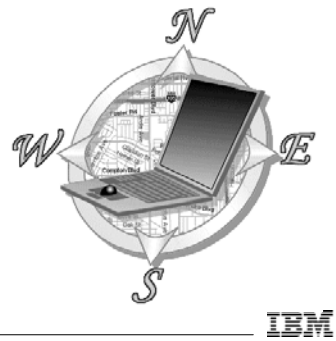
IBM

State machines should be used when they help to better understand and communicate the analysis and design of the project.

Drawing UML Diagrams

Where Are We?

- ♦ UML Diagrams
- ♦ **Drawing UML Diagrams**

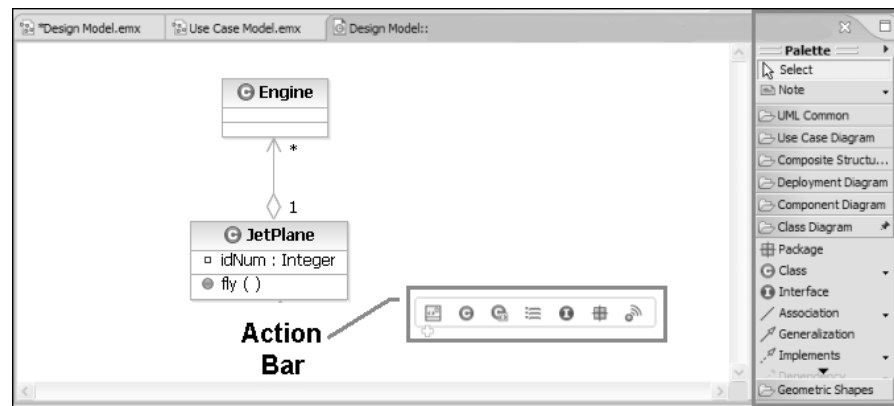


This section examines the diagramming tools in the Rational Modeling Tools **Modeling** Perspective: the diagram editor and the Diagram Navigator view.

Drawing Diagrams in Rational Modeling Tools

Drawing Diagrams in Rational Modeling Tools

1. Use tools from the tool palette.
2. Use the action bar.



19

IBM

The diagram editor window contains both the drawing surface and the palette. You can add shapes to the model by right-clicking the diagram surface or by right-clicking a package in the Model Explorer and then clicking **Add UML** and selecting the appropriate shape.

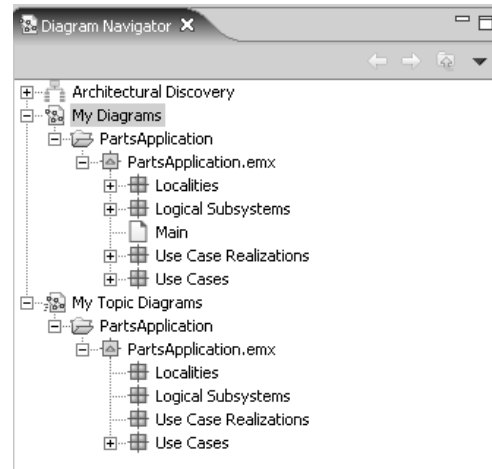
The palette contains a basic set of tools for selecting and annotating shapes in the diagram, along with one or more drawers. You can customize the look and behavior of drawers in the palette, as well as create and customize your own drawers.

Diagram Navigator

Diagram Navigator

Diagrams organized by type:

- Architectural Discovery
 - Generated Topic diagrams
- My Diagrams
- My Topic Diagrams



20



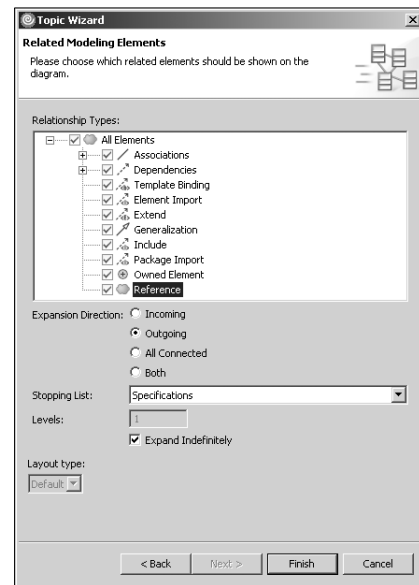
The Diagram Navigator shows the structure of all the diagrams in the workspace. As projects and model structures evolve and become more elaborate, the Diagram Navigator becomes useful for finding and opening the appropriate diagram quickly.

If you were new to an ongoing development project and had to examine an existing set of highly elaborate projects in Rational Software Modeler, you might use the Diagram Navigator to examine the high-level diagrams for each model to get familiar with key abstractions and their relationships.

Topic Diagrams

Topic Diagrams

- ♦ Create Topic diagrams to depict key model elements and their relationships.
- ♦ Topic diagrams:
 - Are created by querying the model
 - Persist in the model
 - Are dynamically updated
 - Are most useful for visualizing code
 - Are used in architectural discovery



21



Rational Modeling Tools can generate and arrange topic diagrams for you automatically. These diagrams are useful when working in large models that contain many elements with complicated relationships. They are also invaluable when trying to help discover the architecture of an application.

Topic diagrams are used for both discovery and as visual documentation because they can be saved in the model. Developing a Topic diagram involves running a query against the existing model contents. You select a key model element and then define what other elements you want to appear in the diagram, based upon the types of relationships they have to the topical element. As the content of the model changes, topic diagrams adjust accordingly.

Topic diagram displays the latest traceability links between all model elements.

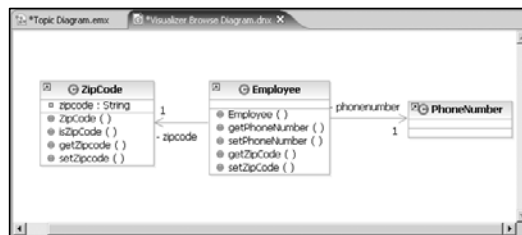
To view model element relationship links in a diagram:

1. In the Model Explorer view, right-click a model element.
2. Click **Add to New Diagram in Model File > Visualize > Topic Diagram**.

Browse Diagrams

Browse Diagrams

- ♦ Browse diagrams can be used to:
 - Show the elements related to the selected element
 - Show the dependencies to the selected element
 - Gain a detailed understanding of the element in focus
- ♦ Browse diagrams are driven by parameters and filters that you control.



22

IBM

Using browse diagrams to view elements in your project can help you to understand the relationships between elements in your code. There are times when you need to gain perspective and understand how an element is used and accessed by other objects.

You can change the level of relationships shown in the browse diagram and use filters to choose specific relationship types. The layout of the browse diagram is based on a radial pattern with the selected element at the center of the dial.

Browse diagrams are not persisted by Rational Software Modeler. However, you can save browse diagrams as a class diagram, .gif, .jpeg, or .bmp file.

To create a browse diagram:

1. In the **Project Explorer**, right-click a Java element.
2. Select **Visualize > Explore in Browse diagram**.

How Many Diagrams to Create?

How Many Diagrams to Create?

- ♦ Depends:
 - You use diagrams to visualize the system from different perspectives.
 - No complex system can be understood in its entirety from one perspective.
- ♦ Model elements will appear on one or more diagrams.
 - For example, a class might appear on one or more class diagrams, be represented in a state machine diagram, and have instances appear on a sequence diagram.
 - Each diagram will provide a different perspective.

23



Good diagrams make the system you are developing understandable and approachable. Choosing the right set of diagrams to model your system forces you to ask the right questions about your system and helps to illuminate the implications of your design.

Demo: Drawing UML Diagrams

Demo: Drawing UML Diagrams

The instructor will now show you how to:

- Add a diagram.
- Add elements to a diagram using the tool palette.
- Add elements or relationships using the action bar.



24

IBM

Watch your instructor demonstrate how to create diagrams in the Rational Modeling Tools.

Lab: Create UML Diagrams

Lab: Create UML Diagrams

- ◆ Given:
 - Maintain Part Information Use-Case Specification
 - Models created in previous lab
- ◆ Complete the following tasks:
 - Create a Use-Case diagram
 - Create an Activity diagram
 - Create a Sequence diagram
 - Populate a Class diagram



25

IBM

Complete Lab 3: Creating UML Diagrams.

Review

Review

- ♦ Which UML diagrams are used to model system structure?
- ♦ What are the two types of interaction diagrams?
- ♦ Describe two ways to add shapes and connectors to diagrams in Rational Software Modeler



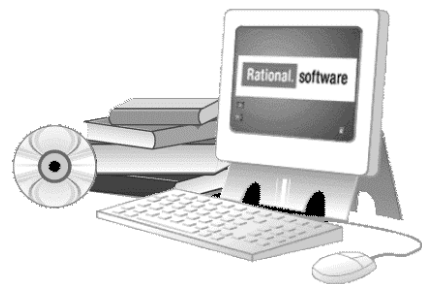
26

IBM

Further Information

Further Information

- ♦ “Model Structure Guidelines for Rational Software Modeler and Rational Software Architect”
- ♦ The Unified Modeling Language Reference Manual



27

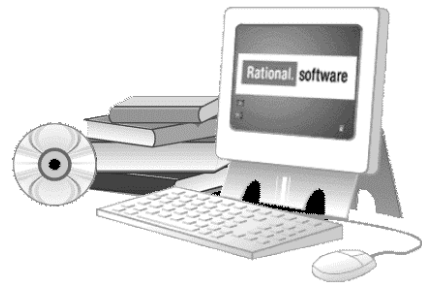


- IBM Rational Software: “Model Structure Guidelines for Rational Software Modeler and Rational Software Architect” (2004 Release).
- James Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual*. 2nd Ed. New York: Pearson Education, 2004.

Related IBM Rational Training

Related IBM Rational Training

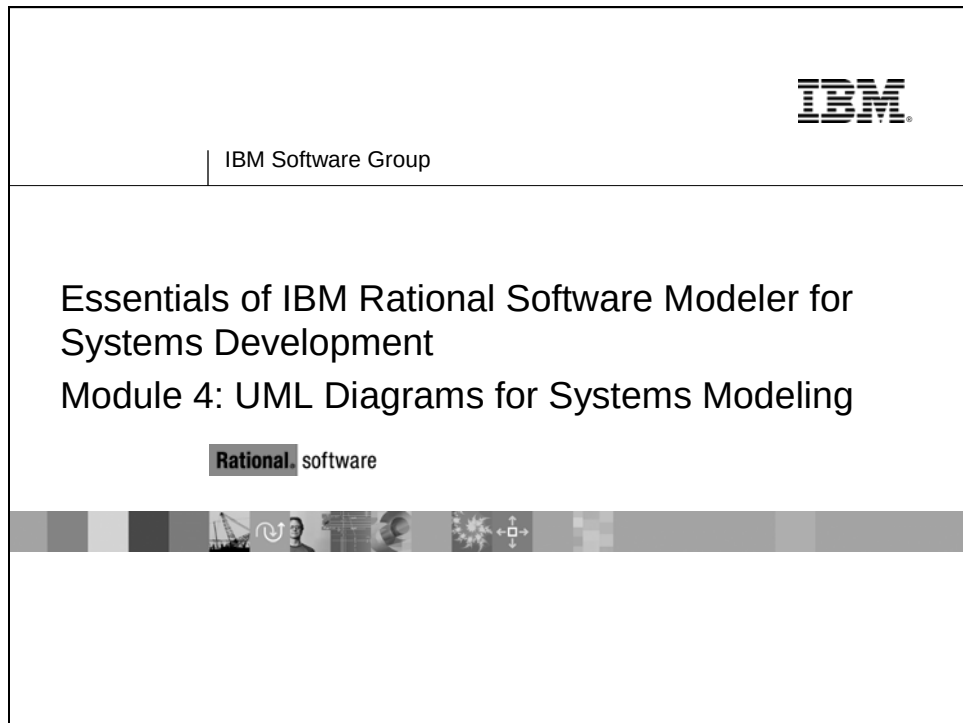
- ♦ DEV275: Essentials of Visual Modeling with UML 2.0 (2004)
- ♦ DEV475: Mastering Object-Oriented Analysis and Design with UML 2.0 (2004)



28



► ► ► **Module 4**
UML Diagrams for Systems Modeling



Topics

Context Diagrams.....	4-3
Sequence Diagrams for Use-Case Flowdown	4-11
Locality Diagrams.....	4-15

UML Diagram for Systems Modeling

UML Diagram for Systems Modeling

♦ Objectives:

- Draw Context, Sequence, and Locality diagrams to model systems
- Apply keywords or stereotypes to extend modeling concepts

2



Context, Sequence, and Locality diagrams are built from diagrams you have just learned:

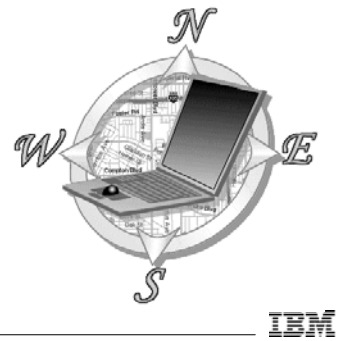
- A context diagram is a specialized class diagram.
- Sequence diagrams are an excellent means of modeling use-case flowdown.
- A Locality diagram is also a specialized Class diagram.

If you have access to the RUP SE Profile, the appropriate stereotypes and icons will be available for you to use. Even if you do not, you can either use the keywords capability of UML or create your own profile for Systems Engineering modeling constructs. Creating a profile will not be covered in this course, but is covered in DEV398: Essential of IBM Rational Software Architect Extensibility.

Context Diagrams

Where Are We?

- ♦ **Context Diagrams**
- ♦ Sequence Diagrams for Use-Case Flowdown
- ♦ Locality Diagrams



3

A Context Diagram

A Context Diagram

- ♦ Models the context of the system
- ♦ Is a specialized class diagram
 - Puts the system as a stereotyped class in the center of the diagram
- ♦ Shows the system under consideration in relationship with its outside actors
- ♦ Shows the data that gets passed between the system and the actors

4



A context diagram is used to explain the system in its broader context. A context diagram shows the scope, boundaries, operations, and input/output (I/O) entities of the system under consideration.

Context Diagram (RUP SE Review)

Context Diagram (RUP SE Review)

- ♦ Model peer systems as classifiers
- ♦ Model operations as operations
- ♦ Add Actors and Input/Output (I/O) entities
- ♦ Show only one level of abstraction

5



Context analysis

The system being developed is always part of a larger enterprise. To specify the system, one needs to understand the activities of the broader enterprise, partition that enterprise into the system and other entities, analyze how the system participates in helping the enterprise provide its operations, and then capture the results of that analysis as the system specification. In this section, you will explore this workflow in more detail.

Note that systems have characteristics of both physical and logical entities. They are logical entities in that they provide operations, pass messages, and interact with other logical entities. Systems are physical in that they have finite limits on their capabilities, and these limits must be considered. In RUP SE, we model systems as stereotyped classifiers but add the relevant physical characteristics as class attributes.

A system specification captures a blackbox description of the system. It establishes the scope and boundaries of the system, the operations it provides, its other attributes, and the things it exchanges with other entities. This information is captured in a system context diagram.

Associations

Associations

- ♦ Represent a structural relationship that connects two classifiers.
- ♦ Are drawn between two classifiers by:
 - Adding the classifier from the palette
 - Using handles on classes

6



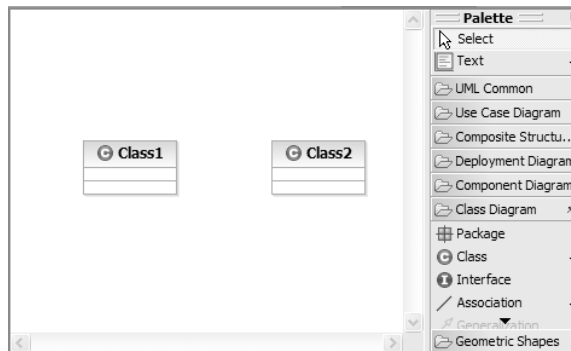
None of the labs so far have demonstrated how to create associations between classifiers (class-like entities). Associations can be drawn between two classifiers either from the palette or by using handles on the classes. In UML modeling, an association is a relationship between two classifiers, such as classes or use cases, that describes the reasons for the relationship and the rules that govern the relationship.

The name of an association describes the nature of the relationship between two classifiers and should be a verb or phrase that you can read in reverse when viewing a two-way association. In the diagram editor, an association appears as a solid line between two classifiers.

Associations - Add from Palette

Associations - Add from Palette

- ◆ Use the palette to draw an association between two classifiers.



7

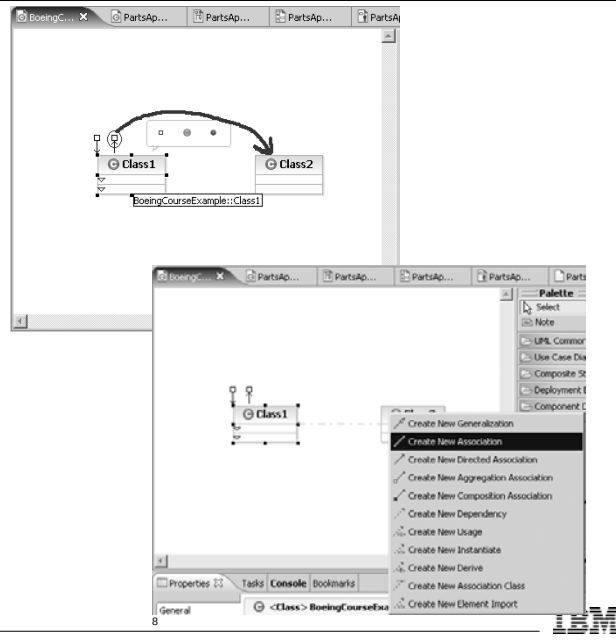


To draw an association between two classifiers using the palette, click on the association in the palette, bring the cursor to the diagram, and drag it from one class to the other. This will add the association between the two classes.

Associations - Use Handles on Classes

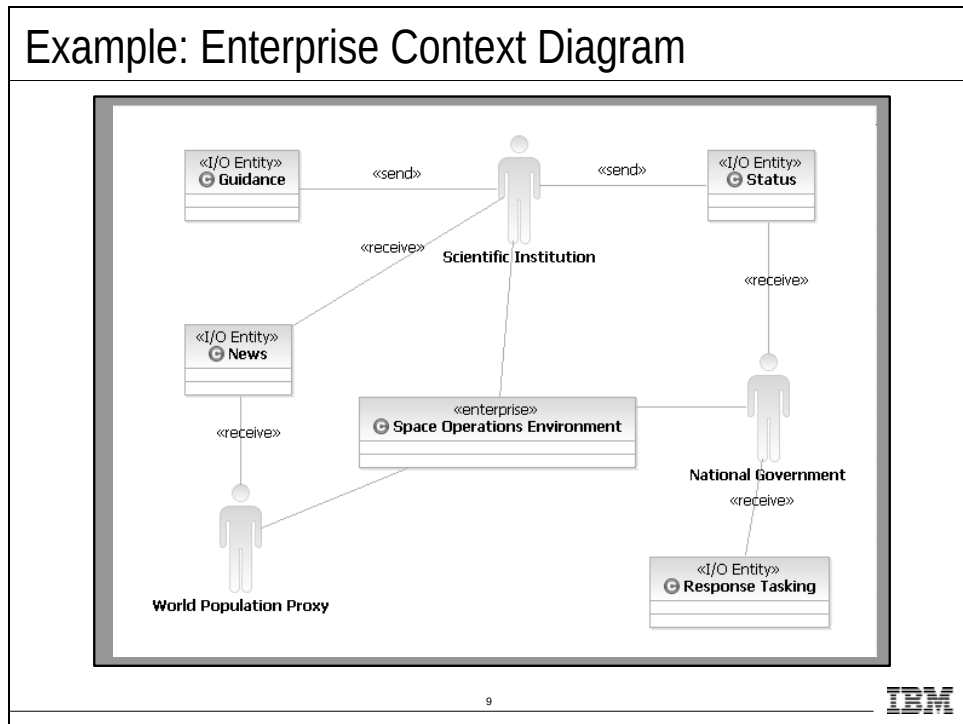
Associations - Use Handles on Classes

- ◆ Draw an association in the diagram by using the handles on classes.



In the diagram, place the cursor over a class, and handles will appear on the class. By dragging the handle to the other class, a menu appears. Choose new association from the menu.

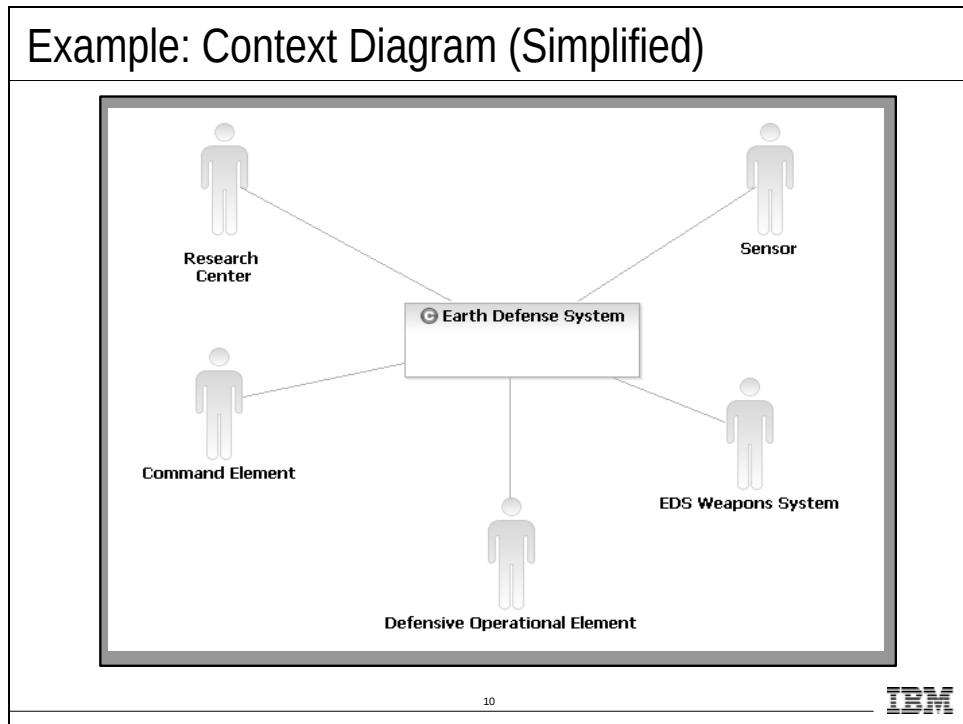
Example: Enterprise Context Diagram



A Context Diagram delineates:

- The interfaces to be realized by the system (in terms of the operations the systems provides, and the associated protocols supported, the state variables and stores that the system realizes, and attributes characterized as Technical Performance Measures)
- The I/O entities that flow between the system and its actors.
- The interfaces required by the system (to be realized by the actors which interact with the system) for correct performance. Often, if the actor represents an existing system with which the system must communicate, these required interfaces will simply reflect constraints imposed by that other system.
- A Context Diagram shows the top-level collaboration between the system and its actors. It is the structural analog to the use-case model for the system.

Example: Context Diagram (Simplified)

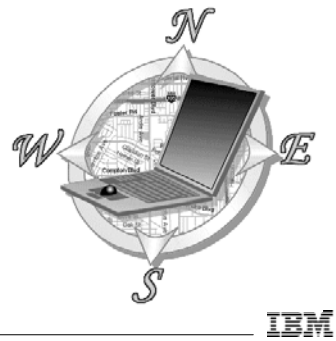


As you evolve and add more detail to a Use-Case Model (discovering the actors, elaborating their interaction), you can create the initial collaboration and illustrate this with a Context Diagram. The Context Diagram can be created as shown, initially with the system interfaces abstracted away. The system is depicted as a top-level subsystem (stereotyped "enterprise"), which eventually realizes several interfaces. Actors and their associations are also shown, again, with no detail initially.

Sequence Diagrams for Use-Case Flowdown

Where Are We?

- ♦ Context Diagrams
- ♦ **Sequence Diagrams for Use-Case Flowdown**
- ♦ Locality Diagrams



11

Sequence Diagrams Model Flowdown

Sequence Diagrams Model Flowdown

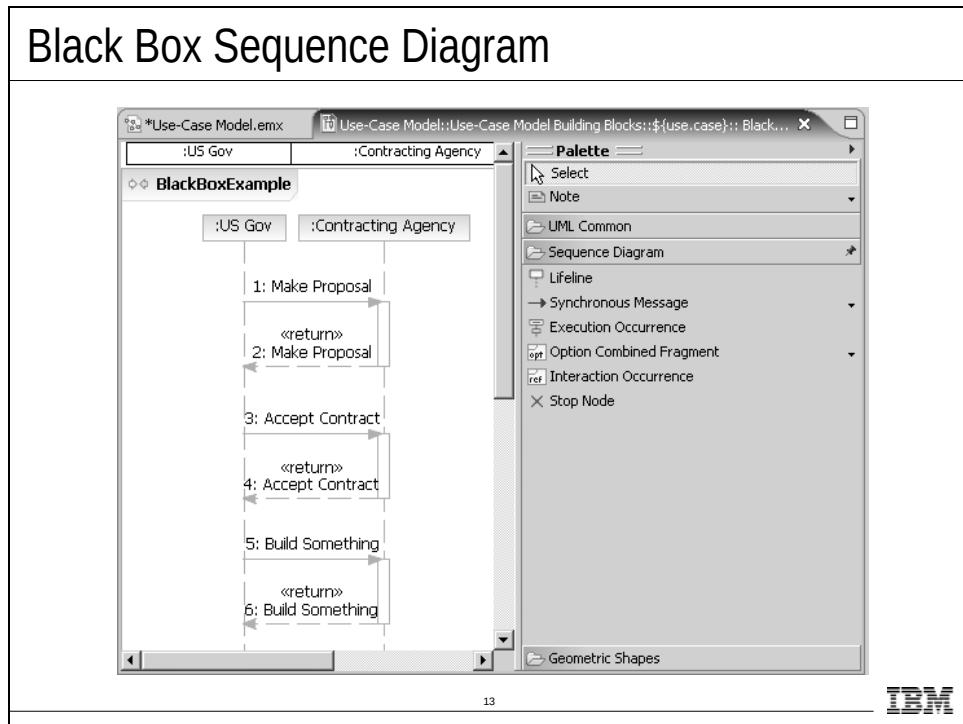
- ♦ “Black box” Sequence diagrams describe interactions between the system and its actors.
- ♦ “White box” Sequence diagrams describe interactions between the component parts of the system and its actors.

12



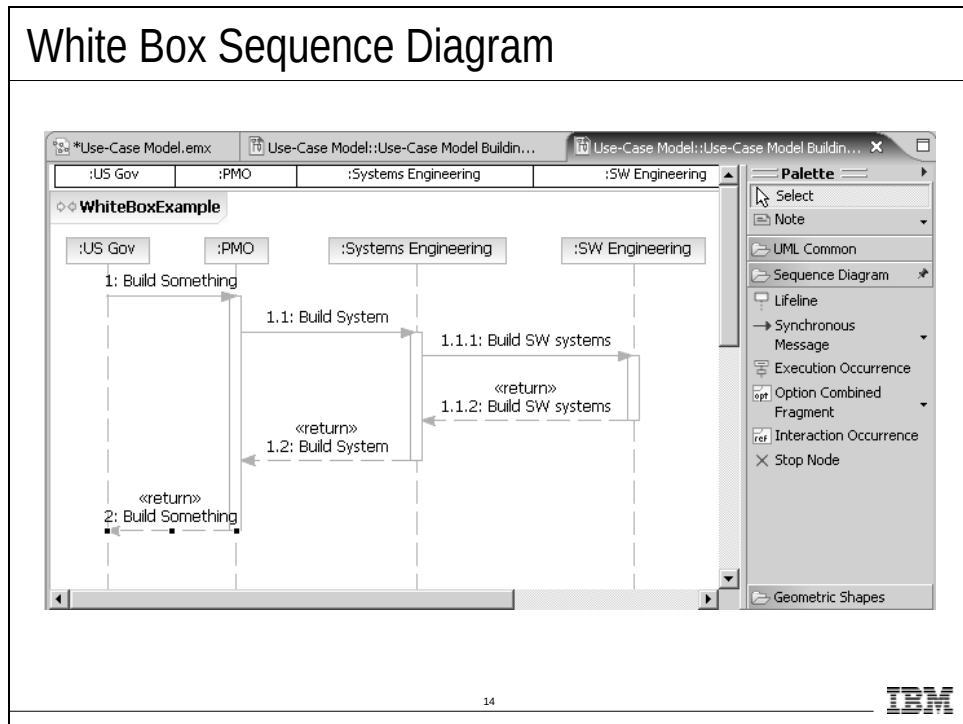
There is no difference in how each of these diagrams is constructed; the only difference is the participants in the collaborations. In black box diagrams, the system under consideration and its actors are the participants. In white box diagrams, the subsystems or component parts of the system and the actors are the participants.

Black Box Sequence Diagram



This slide and the next one are obvious oversimplifications. The point is, at some conceptual level, the government, or some outside funding agency, contacts a Contracting Agency to build something. There is a series of interactions that can be usefully modeled at this level. The final result of this interaction is that the Contracting Agency is requested to build something. The Contracting Agency then provides a set of operations as part of this interaction.

White Box Sequence Diagram

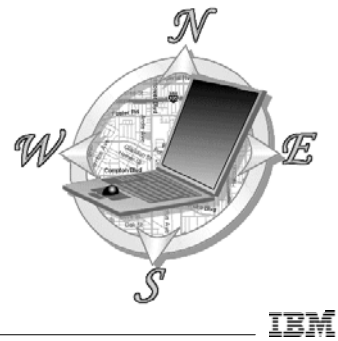


In this case, again a simplification of what actually occurs, we model the operation “Build Something” as a collaboration between the outside actor and the internal “systems” or “subsystems”. Note that an organization can be considered a system.

Locality Diagrams

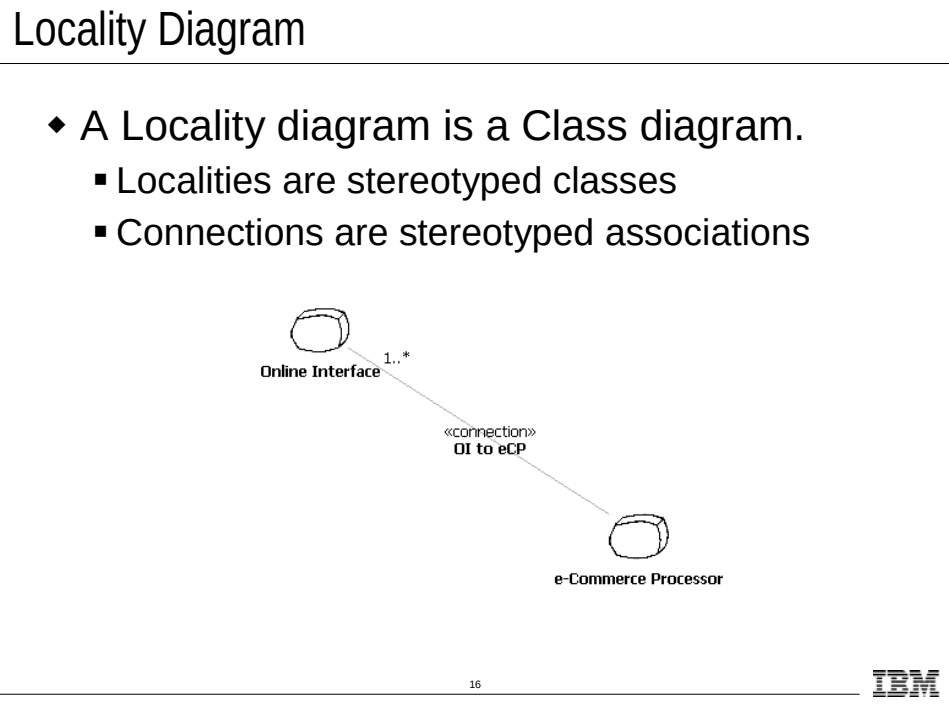
Where Are We?

- ♦ Context Diagrams
- ♦ Sequence Diagrams for Use-Case Flowdown
- ♦ **Locality Diagrams**



15

Locality Diagram



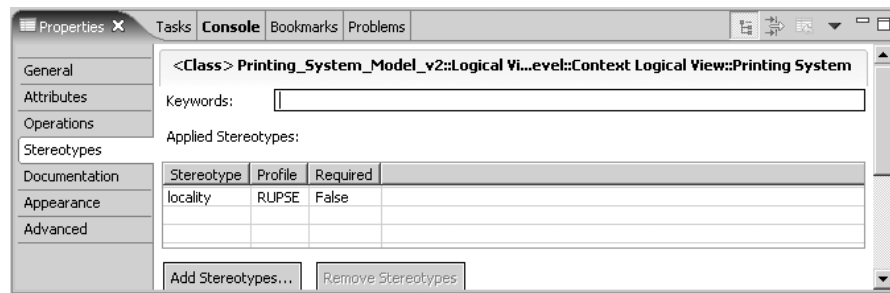
The **locality** model represents the initial, abstract, **physical** partitioning and distribution of the system, and is concerned with the physical resources of the system (nodes, devices, sensors and their physical connections and interfaces, and the physical characteristics of these, for example weight, heat generation, power consumption, vibration, and others).

A **locality** expresses notionally where processing occurs (the semantics of locality implies a tighter grouping of resources) without defining exact geographic location or how the processing capability is to be realized. It is conceivable for very complex, very large systems, that the initial locality model can have localities that decompose to finer-grained localities (just as a subsystem can contain subsystems).

Stereotypes

Stereotypes

- ♦ Applied stereotypes are added in the properties tab.



17



If a RUP SE profile is available, you can add the appropriate stereotypes through the properties viewer. Otherwise, you can use the keyword field to specify RUP SE entities.

Modeling Localities

Modeling Localities

- ◆ Localities can apply at each level of abstraction.
 - Enterprise
 - System
 - Subsystem
- ◆ Localities map to a Physical Model, instead of Logical.
- ◆ A locality is a place where something happens.

18



Modeling Localities (cont.)

Modeling Localities (cont.)

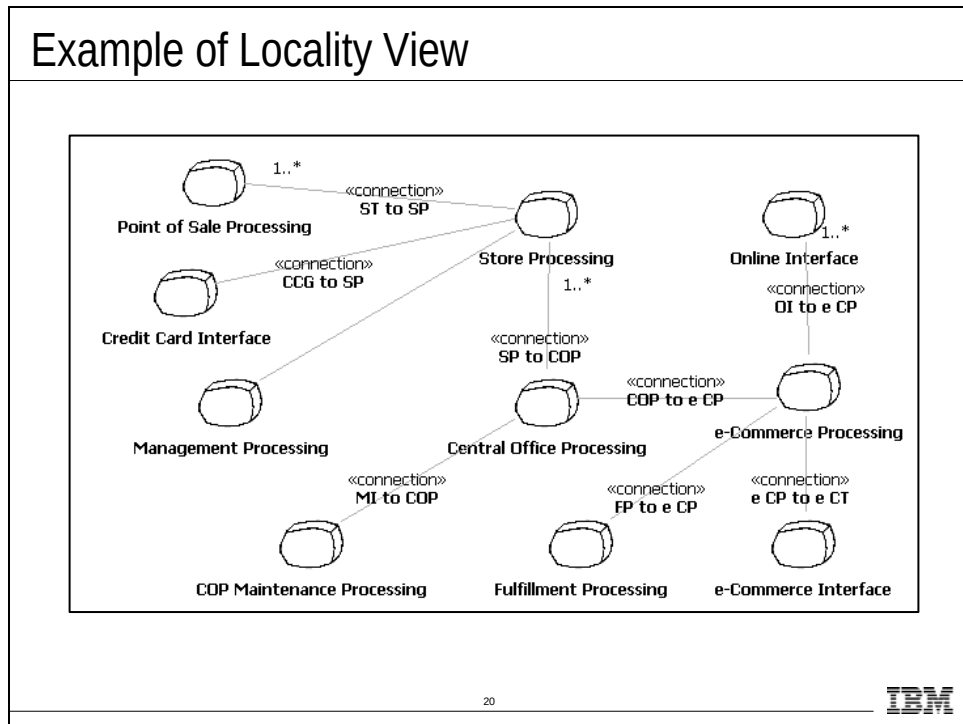
- ♦ Draw physical structure using class diagram.
 - Classes represent physical elements; localities are represented by stereotyped classes.
 - Each class inherits its hosted operations from the interface class containing that operation.
 - Re-factor operations into interfaces to enable re-use among physical elements.
- ♦ A Locality diagram shows localities and their physical interconnections.

19



The locality diagrams show the initial partitioning, how the system processing elements are distributed, and how they are connected.

Example of Locality View



Demo: RUP SE Artifacts

Demo: RUP SE Artifacts

- ♦ The instructor will now show you how to:
 - Create a Context diagram.
 - Create flowdown Sequence diagrams.
 - Create a Locality diagram.



21

IBM

Lab: UML Diagrams for System Modeling

Lab: UML Diagrams for System Modeling

- ♦ Given:
 - RUP SE Plugin for Rational Software Modeler
- ♦ Use RSM to produce:
 - A Context Diagram
 - Two Sequence Diagrams to illustrate flowdown of one enterprise operation
 - A Locality diagram



22

IBM

Use Lab 4: UML Diagrams for Systems Modeling to complete this exercise.

Review

Review

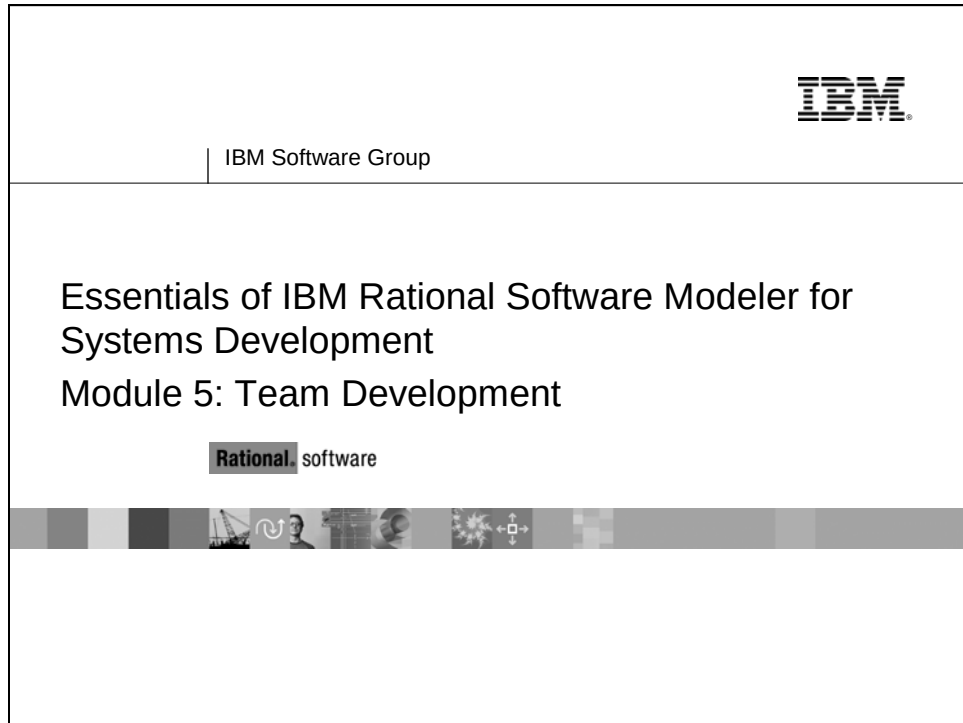
- ♦ What kind of diagram is a Context Diagram?
- ♦ What kind of diagram is used to model use-case flowdown?
- ♦ What kind of diagram is a Locality diagram?



23

IBM

► ► ► **Module 5**
Team Development



Topics

Configuration Management in Rational Software Modeler.....	5-4
Compare and Merge	5-9
Model Publishing	5-15

Team Development

Team Development

Objectives:

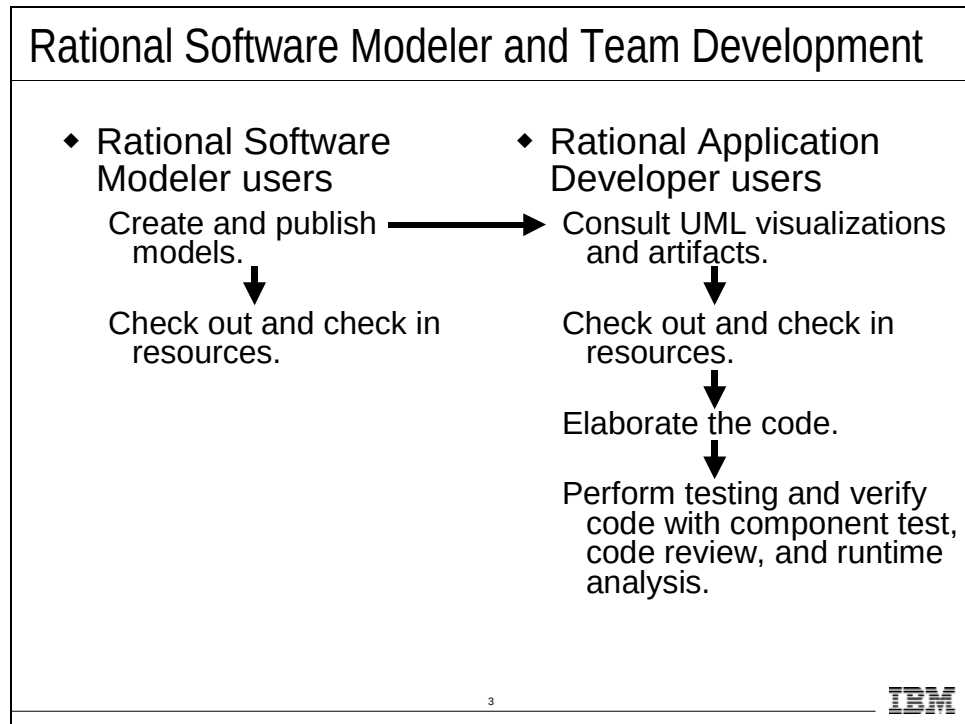
- Describe team development best practices with Rational Software Modeler.
- Compare and merge diagrams in Rational Software Modeler.
- Publish and share diagrams with other team members.

2



This module introduces team development with Rational Software Modeler, including configuration management features and best practices.

Rational Software Modeler and Team Development



The following features facilitate collaboration between architects and developers:

Transformation

The model transformation feature allows architects to create basic code structures from high-level UML models, which can be used as a starting point for developers.

Code Review

The code review feature is used to analyze code against the project's architectural guidelines. Code review includes:

- Structural anti-pattern detection to assist in Java refactoring.
- Structural rules (including user-defined rules) to enforce architectural control of Java code.

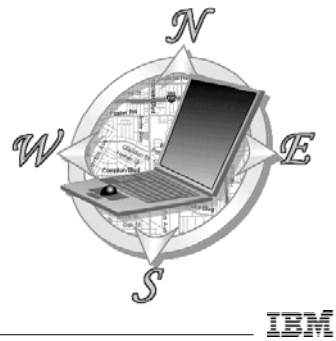
Web Publisher

While creating detailed implementation code, the developer can refer to Web-published models and UML-injected Javadocs, which Software Modeler can generate automatically.

Configuration Management in Rational Software Modeler

Where Are We?

- ♦ **Configuration Management in Rational Software Modeler**
- ♦ Compare and Merge
- ♦ Model Publishing



4

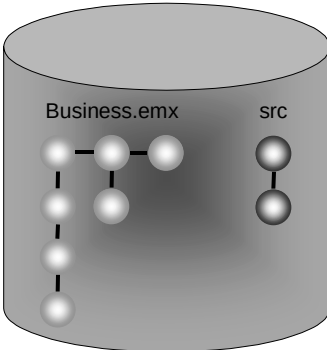
IBM

This section provides an overview of configuration management issues.

Configuration Management (CM)

Configuration Management (CM)

- ♦ CM allows change in software assets to occur in a structured, controlled, and repeatable fashion.
- ♦ A CM tool can:
 - Give team members simultaneous access to models.
 - Control who can update different model elements.
 - Help introduce changes in a controlled manner.
 - Maintain the evolutionary history of a model and its elements.
- ♦ A CM process defines how tools will be used to manage change in a project.



Repository

5


Configuration management (CM) is a software engineering discipline that comprises the tools and techniques (processes or methodology) that a company uses to manage change to its software assets. The goal of successful CM is to allow change in a structured, controlled, and repeatable fashion.

Successful CM is achieved by applying both tools and processes to a software development project.

CM tools are software tools that automate and facilitate CM activities.

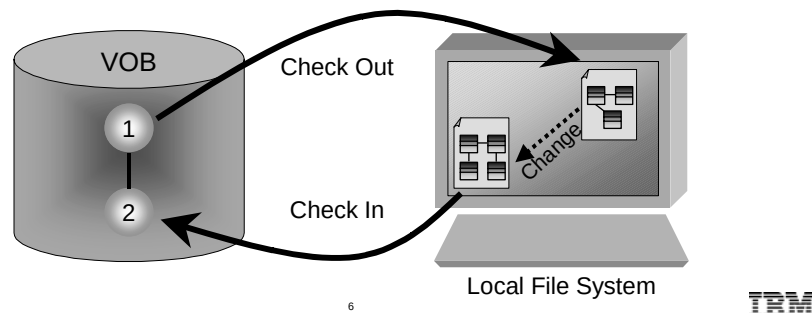
A CM process reflects how CM is performed on your project; specifically, how a CM tool is applied to accomplish a task. The key to a good process is in customizing it to be as simple as possible, following standard industry practices.

Another important strategy for team development is to facilitate the reuse and sharing of existing project assets, such as patterns and components. Remember that changes to reusable assets must be controlled the way changes to any other project artifacts are controlled.

Configuration Management with Rational ClearCase

Configuration Management with Rational ClearCase

- ♦ Use IBM Rational ClearCase to:
 - Add model, diagram, and code files to source control.
 - Check out and check in files.
 - Merge changes automatically.
 - Maintain a history of changes to a file.
 - Compare and merge versions of a file.



IBM® Rational® ClearCase® provides software configuration management for medium to large-sized teams. ClearCase can manage all the artifacts in the development process from requirements documents and design models to code and tests. ClearCase supports:

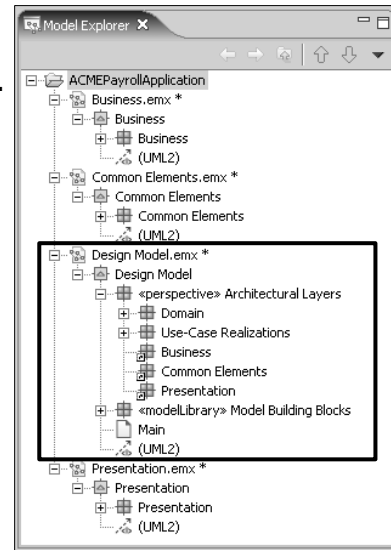
- Parallel development
- Advanced workspace management, including dynamic views
- Build management features, including mainframe connectors

ClearCase integrates with Rational Software Modeler, including the compare and merge features. When team members work with version-controlled artifacts, they must frequently compare file versions to understand their history and development. Team members might need to merge files because a merge result generally contains all the work that multiple team members complete in parallel.

SCM Best Practices: Model Partitioning

SCM Best Practices: Model Partitioning

- ◆ Partition the model to avoid unnecessary merges.
- ◆ Factors to consider when deciding how to partition a model:
 - Stabilize abstraction levels
 - Minimize dependencies between models
 - Establish ownership policies
 - Avoid broken references



7



While the compare and merge features in Software Modeler are advanced and easy to use, model merge scenarios can become complex and prone to user error if the models involved are needlessly complicated and disorganized. To optimize your models for configuration management and team development, you should observe the following principles:

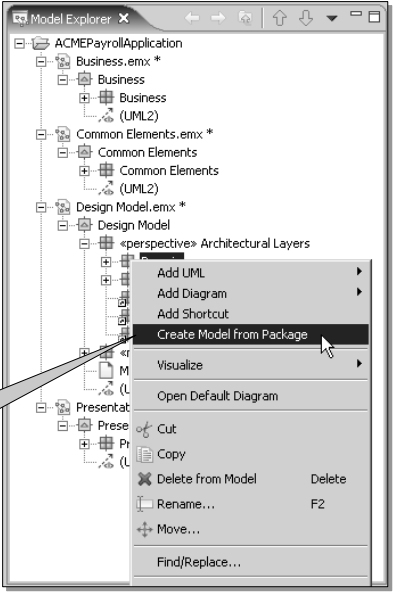
- **Stabilize abstraction levels:** Do not partition the model until the top-level subsystems are clearly defined. After the subsystems are mature and stable, you can separate them to enable parallel development and to improve the speed with which the model opens. When an individual subsystem's contents stabilize, you can then separate the subsystem from the model.
- **Minimize dependencies between models:** By minimizing dependencies between models, you can reduce the likelihood that a change in one model will affect another. If you allow dependencies between models, widespread conflicts can occur that must be resolved with out of context model merges (which are very challenging).
- **Establish ownership policies:** Assigning different model files to different team members can help avoid conflicts and help speed integration when you deliver the model to a shared work area. You should establish the size and scope of each model so that a single person can work on it.
- **Avoid broken references:** Whenever you move a model partition outside of your configuration management system, you break that model file's references. You have to repair these broken references whenever you reintegrate the model back into the CM system. It can be time-consuming and error-prone to resolve a large number of references in a complicated model.

Model Partitioning

Model Partitioning

- ♦ To partition a model, you can:
 - Create new models from packages.
 - Copy packages from partitions back into the main model.

Creating a model from a package automatically leaves a "shortcut" reference to the new model where the package used to be.



The screenshot shows the Model Explorer window with a tree view of a project named 'ACMEPayrollApplication'. The tree includes packages like 'Business', 'Common Elements', and 'Design Model'. A right-click context menu is open over the 'Design Model' package, with the option 'Create Model from Package' highlighted. Other options include 'Add UML', 'Add Diagram', 'Add Shortcut', 'Visualize', 'Open Default Diagram', 'Cut', 'Copy', 'Delete from Model', 'Rename...', 'Move...', and 'Find/Replace...'. A callout box points to the 'Create Model from Package' option with the text: 'Creating a model from a package automatically leaves a "shortcut" reference to the new model where the package used to be.'

8

IBM

You can refactor an existing model file into several, smaller model files in Software Modeler.

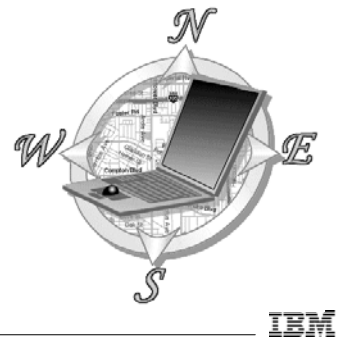
To separate a package from a model into a new model file in the Model Explorer, right-click the package and then click **Create Model From Package**.

To reassemble the model after it has been partitioned, you must copy and paste the packages back into the main model.

Compare and Merge

Where Are We?

- ◆ Configuration Management in Rational Software Modeler
- ◆ **Compare and Merge**
- ◆ Model Publishing



9

IBM

This section shows how to compare and merge versions of a model.

Compare and Merge Models

Compare and Merge Models

- ♦ Rational Software Modeler allows merging of model and diagram files using the compare and merge utility.
 - Compare models to identify changes between model versions.
 - Merge models when:
 - Parallel development occurs.
 - Alternative approaches are explored.
 - Circumstances dictate.
 - Avoid situations that require frequent merging.

10



You can compare two or three models with a common ancestor to find the differences among them outside a source-controlled environment or within a team environment, using a configuration management system, such as IBM Rational ClearCase.

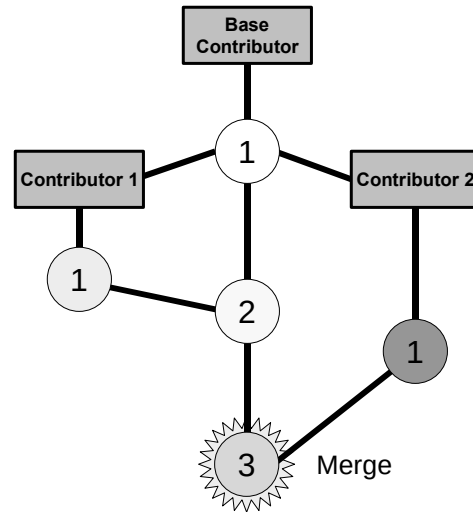
The compare feature can be used to compare versions of models just to see what has changed between versions as part of everyday modeling work, particularly in collaborative settings. Compare and merge becomes essential in more complicated team development settings, such as parallel development.

Compare and merge scenarios can often become very complicated and difficult to resolve if not carefully controlled. When working with large and complicated models, compare and merge should be avoided except when absolutely necessary.

Merging Models

Merging Models

- ♦ Begin with a base contributor, the “common ancestor” of the models you want to merge.
- ♦ Have up to three contributors (modified versions of the base model) in a merge session.



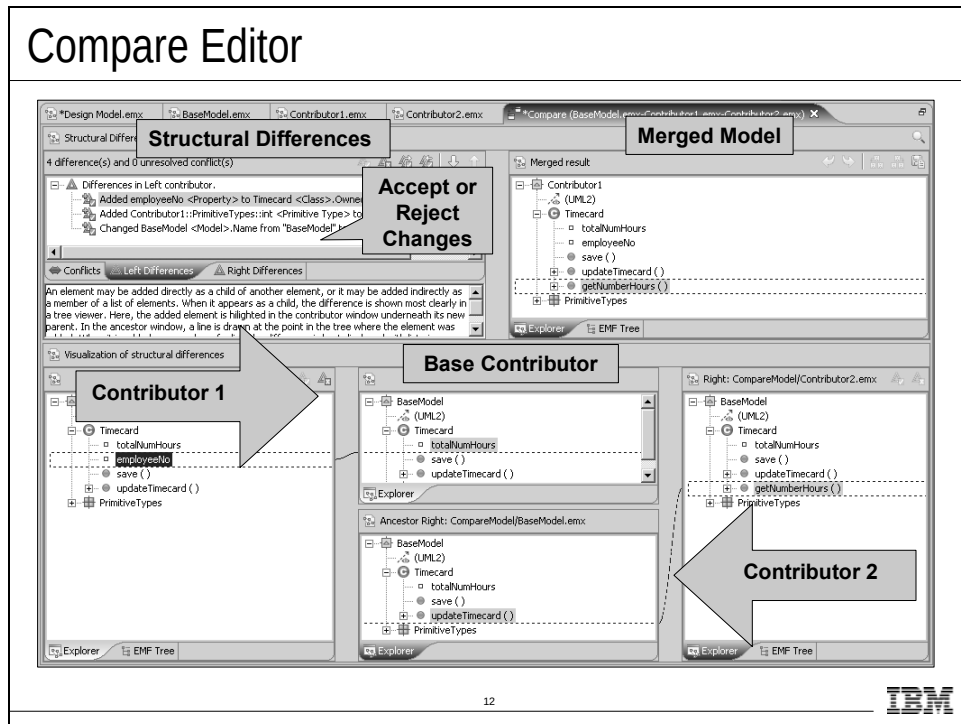
11



The slide shows the simplest and most common compare and merge scenario:

- You and a colleague work on copies (Contributor 1 and Contributor 2) of a model file (Base Contributor) that is under source control.
- You both make changes to your copies.
- Your colleague checks in changes (creating Base Contributor version 2).
- You try to check in changes.
- The SCM system detects that the latest version in the repository (BC version 2) is not the version you checked out and modified (BC version 1).
- The SCM system detects that your version cannot replace the version in the repository, or else you lose your colleague’s changes.
- The SCM system attempts to merge your changes into the latest version (BC version 3) in the repository.

Compare Editor



Given multiple versions of a model that were modified in parallel, you can merge the models by starting a merge session and using the Compare editor. After you resolve all conflicts among the contributors, you can save the final merged model as a new version. When you compare two versions of a model, their differences are displayed in the Differences editor. When you compare three models, two sets of differences are displayed:

- Differences between one version (the left contributor) and the common ancestor (the base contributor).
- Differences between the other version (the right contributor) and the common ancestor.

When working with a version control system, such as ClearCase, a merge session can start automatically when a user checks in a file under version control. After merging models, saving the merged model, and closing the session, the process that initiated the check-in process continues.

Automatic merges

During an automatic merge, all differences and trivial conflicts are merged automatically. If the software automatically makes a resolution, a resolution icon is displayed beside the resolved difference in the Structural Differences view. The software also generates a merged model for these resolutions.

If the merged model contains conflicts, a conflict icon is displayed beside each conflicting element in the Structural Differences view.

Manual merges

During a manual merge, you resolve the differences or conflicts. You can right-click a difference or conflict and then accept or reject it. You can also accept or reject groups of differences or conflicts.

You can start a manual merge session in the following ways:

- Merge models in a ClearCase version or history interface
- Compare models in an Eclipse resource view that has one or more modifiable resources

Demo: Comparing and Merging Models

Demo: Comparing and Merging Models

- ♦ The instructor will now show you how to:
 - Compare two contributors.
 - Merge three contributors.



13

IBM

Watch your instructor demonstrate how to merge models.

Lab: Team Development

Lab: Team Development

- ♦ Given:
 - Base model
 - Two contributor models
- ♦ Complete the following tasks:
 - Compare two contributors
 - Merge three contributors



14

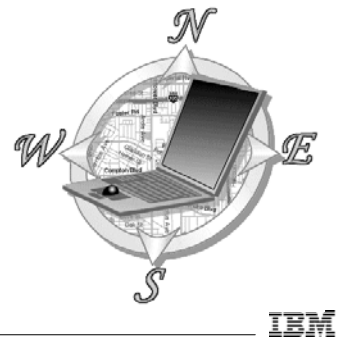
IBM

Complete Lab 5: Team Development.

Model Publishing

Where Are We?

- ♦ Configuration Management in Rational Software Modeler
- ♦ Compare and Merge
- ♦ **Model Publishing**



15

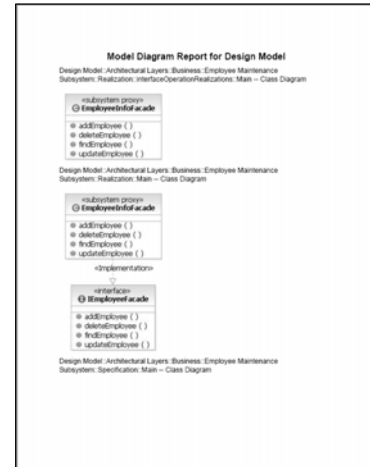
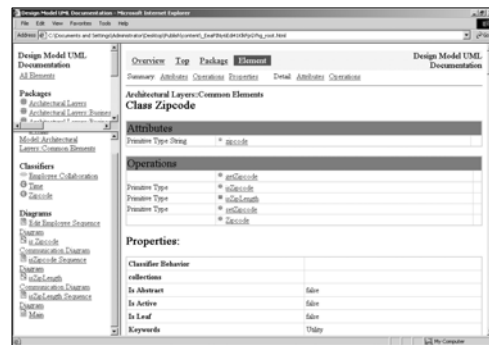
IBM

This section discusses the model Web publishing features of Software Modeler. Publishing is important for sharing models and diagrams with team members, collaborators, and customers.

Model Publishing

Model Publishing

- ◆ Publish the entire model to HTML.
- ◆ Publish reports to PDF.
 - Model diagram report
 - Sample UML metric report



16



When you publish a model as a Web site, you indicate the scope of data to include, how much about that data to include, and where you want to save the published model.

Each published model is a separate, wholly contained entity with its own root page. When you publish, the root page has a link to each included model.

When and Why to Publish Models

When and Why to Publish Models

- ◆ Publish models as an aid for manual model and architecture review.
 - Publish an established API or framework
- ◆ Work with developers using Rational Application Developer.
- ◆ Share models with customers and partners who are not Rational Software Modeler users.

17



Publishing extracts model information and uses it to convert model elements, including diagrams, classes, packages, relationships, attributes, and operations, to images and model information to HTML for viewing outside the modeling tool.

In the published model, hypertext links let viewers navigate the model much as they would within Software Modeler.

You can publish successive iterations of an evolving model for review or to share information with team members or customers. Another potential use is to publish documentation for a frozen API or framework.

Demo: Model Publishing

Demo: Model Publishing

- ◆ The instructor will now show you how to:
 - Publish a model to HTML
 - Explore the published model
 - Start page
 - Packages
 - Elements
 - Diagrams



18

IBM

Watch your instructor publish a model.

Review

Review

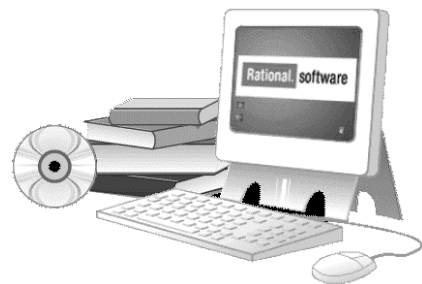
- ♦ Why should you partition model files?
- ♦ What are the four best practices for model partitioning?
- ♦ Name some examples of when you might Web-publish a model.



Further Information

Further Information

- ♦ Rational Unified Process topics
- ♦ Software Configuration Management Strategies and Rational ClearCase



20

IBM

- Rational Unified Process, developer topics:
 - **Designer > Whitepapers > Developing Large-Scale Systems with the Rational Unified Process**
 - **Designer > Implementation > Workflow**
 - **Designer > Configuration and Change Management: Overview**
 - **Designer > Tool Mentors > Rational ClearCase**
- Brian A. White, *Software Configuration Management Strategies and Rational ClearCase: A Practical Introduction*. Boston: Addison-Wesley, 2000.