# **Using Rational Rose UML CASE Tool**

Richard Felsinger, <u>www.felsinger.com</u>, 960 Scottland Dr, Mt Pleasant, SC 29464 <u>dick@felsinger.com</u> or dfelsinger@home.com 843-881-3648 2/10/00 Comprehensive UML Model Building and Java/C++/VB Development and Training Available

The purpose of this tutorial is to present step-by-step instructions and an example to use the Rational Rose UML CASE tool to create a complete UML model. This tutorial is intended for new users of to be able to effectively use and evaluate the Rational Rose tool other UML CASE tools. As a new user, we suggest you do the following:

First: create the examples of the diagrams and reports shown below using the Rational Rose UML CASE tool - www.rational.com.

Second: create a similar example in your own problem domain, e.g. an Order System, Product System, Customer System, etc. using the Rational Rose UML CASE tool.

Third (if required), create the examples for other UML CASE tools to fully compare various tool capabilities.

Our objective is to create a complete UML model of a system that is correct, reliable, understandable, extensible, and reusable. Model should incorporate reusable classes, components, and patterns. A simplified Bank Account UML model is provided.

### **Getting Started with Rational Rose**

Launch Rose; Select File New; Select File SaveAs - BusinessModel. Later select File New SaveAs Requirements-AnalysisModel; Note when Requirements-Analysis is complete then SaveAs Design-ImplementationModel; Select View to check Status Bar, Browser, Documentation, As Unified; Select Tools - Options - Diagram to check Stereotype Display - Label and check all Show/Display items except 3-Tier Diagram; You may enter documentation information in the Documentation Window for each Browser element; You may add text boxes "ABC" on diagrams; You may add Notes on diagrams; Use File - Delete from Model to delete incorrect elements; After Check Model, select Window - Log to view the Log File; You may select Report - Documentation Report; Select Help - Rational Rose Help Topics.

**Creating Rose Use Case Diagram:** In Browser Window select Use Case View, Rename Main to "Use Case Diagram", Double-click on "Use Case Diagram" to display the use case diagram, Place actors and use cases on the diagram, Select a relationship to connect elements.



**Creating Rose Class Diagram:** In Browser Window select Logical View; Rename Main to "Class Diagram"; Double-click on "Class Diagram" to display the class diagram; Place classes on the diagram; To enter an attribute, select a class then press Right Mouse Button and select New Attribute; To enter an operation, select a class then press Right Mouse Button and select New Operation; To place a generalization relationship, select the generalization arrow from the Toolbar then drag from the subclass to the superclass; To place a realizes relationship, select the realizes arrow from the Toolbar then drag from the larger/superior class to the smaller/subordinate class; Right Click the Mouse Button on the relationship line closest to the smaller/subordinate class to enter the role name and role multiplicity. To place an aggregation/composition relationship, select the Mouse Button on the relationship ine closest to the part class to enter the role name and role multiplicity. To place an aggregate-whole class to the smaller/part class; Right Click the Mouse Button on the relationship, select the aggregation diamond symbol from the Toolbar then drag from the larger/superior on the relationship line closest to the part class to enter the role name and role multiplicity; To place a dependency relationship, select the dependency arrow from the Toolbar then drag from the drag from the larger/superior class to the smaller/subordinate class; Select Tools - Check Model; Select Report - Show Usage/Show Instances/Show Access Violations; Select File - Save.

(c) Copyright Richard Felsinger 2001 Reproduction authorized for technical review or non-commercial education purposes.



**Creating Rose Sequence Diagram**: In Browser Window select Logical View; Select Browse - Interaction Diagram - Logical View - <New>; Select Sequence Diagram; Enter Diagram Name; Place objects on the diagram to represent objects of classes; Double-click the object and select the class name from the pull-down list; Select Rose Object Message symbol and drag between objects; Double-click on the message to select the operation. Select Tools - Check Model; Select Report – Show Unresolved Objects/Show Unresolved Messages; Select File - Save.



# **Using the Unified Process Lifecycle Phases**

This tutorial uses the Unified Software Development Process. See *The Unified Software Development Process* by Ivar Jacobson, Grady Booch, and James Rumbaugh, and *The Rational Unified Process An Introduction Second Edition* by Krutchen. The Unified Process Lifecycle phases are graphically shown below.

# System/Subsystem/Component

# I – Requirements

**Requirements Statement/Product Capabilities** 

Γ	

Use Case Diagram for All Use Cases



Sequence Diagram for Each Use Case Scenario



Activity Diagram for Each Use Case All Scenarios



# Package/Class/Object



# **IV - Implementation**

Processing Environment HW & SW Component Diagram & Deployment Diagram

# III - Design

Processing Environment HW & SW Updated Class/Package/Sequence/ Collaboration/Activity/State Diagrams



V & VI Construction/Testing



The various models in the Unified Process are summarized in the table below.

	Business	Requiremen	Analysis	Design	Implementatio	Testing
	Model	ts Model	Model	Model	n Madal	Model
Key UML Elements	Business Processes (Use Cases), Business Domain Objects	System, Actor, Use Case, Interaction	Business Package, Class, Object, Message	HW & SW Configuration, Package, Class Object, Message	Component, Node, Code	Test Plan and Test Cases
Key Concern	Model Business	Model System as a Black Box	Model Business Elements in the Problem Domain with no implementation details	Update Analysis Diagrams/Specificati ons for a specific implementation, e.g. HW & SW Configuration.	Model physical elements for the distributed environment; Code to meet all requirements	Unit (Class/Oper ation) Tests, Integration/ Overall System Tests
Objective - weak coupling - strong cohesion among elements	Sufficient Business/ Enterprise information	Sufficient information on all use cases/scenario s. All increments/iter ations planned.	Simplest Business/Proble m Domain Model to meet requirements	Sufficient information to generate maximum code or manually code	Optimum Component Architecture - Network friendly; Code that meets all requirements	Sufficient Testing that code meets all requirement s
Static/Structur al Diagrams	Business Domain Objects	Block Diagram and Use Case Diagram Showing Actors	Package/Class Diagram	Package/Class Diagram	Component and Deployment Diagrams; Reversed Class Diagrams	-

### Models in the Unified Process

Dynamic/Time Based Diagrams	Business Processes (Use Cases)	Use Case Diagram, Sequence Diagram for each use case scenario, Activity Diagram for each use case	Sequence Diagram for each use case scenario, Statechart for each state-based class, Activity Diagram for each complex operation	Sequence Diagram for each use case scenario, Statechart for each state-based class, Activity Diagram for each complex operation	Optionally update sequence diagrams showing distributed messages	
Key Team Players	Business Analysts	Business Analysts	Business Analyst	Developer	Architect, Developer	Developer/ Tester
Model Sign-off	Project Manager, Architect, Client	Project Manager, Architect, Client	Project Manager, Architect, Client	Project Manager, Architect	Project Manager, Architect	Project Manager, Architect, Client for Acceptance

The suggested the tasks/exercises the create a UML model based upon the Unified Process are:

#### 0 - Business Modeling - Enterprise Level

Review the Business Model: Vision Document, Organization Chart, Business Events and Processes (Use Cases), Business Actors, Workers, and Entities (Domain Model), Business Rules Catalog, Business Interfaces (Set of Operations), Business Patterns, Business Systems Architecture - Component Diagram, Glossary. See *The Rational Unified Process An Introduction* by Krutchen and *Business Modeling with UML* by Eriksson and Penker.

#### I - Requirements - System/Subsystem/Component Level (Implementation Language Independent)

1 - **Requirements** - Review the Requirements Statement, System Drawing, and System Block Diagram (Customer Provided). Optionally, create a System Collaboration Diagram (Context Diagram) showing object/data inputs and outputs to set the system boundary. Optionally, create the Requirements Traceability Table listing Requirement Name, Number, Reference, Use Case, UML Element, Test Case, Description, Responsibility, etc. Optionally, review management plans, schedules, risks, naming/coding standards, methodology plans - UML process/stereotypes/properties/constraints.

2 - All Use Cases - Create the Use Case Diagram showing all use cases. Optionally, show use case relationships (includes, extends, generalization) and/or hierarchical use cases (high level to detailed). Identify the use case increments and iterations.

3 - Each Use Case - Create a Use Case Specification for each use case stating use case name, trigger, inputs, outputs, precondition/exception, postcondition/exception, basic and alternative scenarios (optimistic to pessimistic), business rules. Optionally, create a Sequence Diagram for each use case scenario. Optionally, create input/output forms and test case for each use case.

4 - Each Use Case All Scenarios - Create an Activity Diagram for each use case showing all use case scenarios (optimistic to pessimistic). Optionally, show all scenarios/paths for a use case in text, flow chart, or other diagram.

5 - **Information -** Create the **Product Capabilities** listing non-functional requirements: usability (reliability, performance, security, human factors), generality (portability, compatibility), timing, space, memory, etc. Optionally create a semantic data model (High Order Concept Model (HOCM)) showing all major elements/concepts inside and outside the system to be the basis for Class/Package Diagram.

#### II - Analysis - Class and Object Level (Implementation Language Independent)

6 - Classes and Packages - Create the list of candidate classes, CRC Cards (Class Responsibility Collaboration), Class Diagram, and Package Diagram. Optionally create an Object Diagram showing attribute values.

7 - Objects and Messages - Create a Sequence Diagram for each use case - optimistic scenario. Optionally, create a Sequence

Diagram for all other use case scenarios. Alternatively, create a Collaboration Diagram for each use case scenario.

8 - States and Transitions - Create a State Diagram for each state-based class showing states, events, conditions, and actions.

9 - **Operations** - Create an **Operation Specification** showing preconditions, transformations, postconditions, and exceptions for each complex operation. Optionally, create an **Activity Diagram** for each complex operation showing the sequence of activity states, conditions, and actions.

#### III - Design - Class and Object Level for a Specific Processing Environment

10 - **Processing Environment** - Create the processing environment consisting of the planned Implementation H/W and S/W Configuration List: operating system, language, class libraries, components, GUI, distribution - object request broker, persistent data storage, etc. Optionally, list potential patterns, component standard (Active X, Java Bean, CORBA), naming conventions, coding standards, code generation scripts, tools (CASE, compiler, configuration management, testing, etc).

11 - **Updated Analysis Diagrams and Specifications** - Update all diagrams and specifications to add detail for the Processing Environment including data types, visibility, parameters/returns, support classes, operation detail (precondition/exception, transformation, postcondition/exception), etc. Optionally provide implementation of patterns, e.g. polymorphic operations, exceptions (exception superclass/subclasses), threads, data access, transactions, security, message queuing, etc. Goal: diagrams and specifications provide adequate information for manual coding or code generation.

#### IV - Implementation - System/Subsystem/Component Level for a Specific Processing Environment

12 - **Implementation Processing Environment, Component Standard and Component Patterns** - Update the Processing Environment to show provided components (GUI, data access, transactions, distribution, message queuing, security, etc), component standard (Active X, Enterprise Java Bean, CORBA), and component to component patterns, e.g. small single operation component, class based component, session per user - entity component, package based component, large multi-package component.

13 - **Implementation Components** - Create the **Component Diagram** showing all required components and files with the dependency relationship, e.g. .EXE, DLL, .OCX, .LIB, .TXT, .HLP, etc. Optionally show interfaces (lollipops) and create an **interface diagram** showing exposed operations. Optionally show IDL (Interface Definition Language) code, e.g. CORBA IDL, Microsoft IDL, Java.

14 - **Implementation Nodes (Processors and Devices) -** Create the **Deployment Diagram** showing all required processors, devices, and other equipment, e.g. client network computer, Windows PC, NT Server, Transaction Server, Web Server, Mail Server, Fax, Printer, Network, etc.

#### **V** - Construction

15 - Coding Standards and Code Generation Scripts - Update coding standards and code generation scripts. Coding standards list sample code showing code for all major UML elements and relationships and policy for inheritance, interfaces, exceptions, threads, etc.

#### 16 - Code Each Component and Reverse Engineer Diagrams

#### VI - Testing

17 - **Testing Plan** - Update the Testing Plan to list test cases (name, input, output, conditions), test procedures (step by step instructions for each test case), test components (drivers, harnesses, scripts).

18 - **Tests** - Conduct tests, e.g. class/operation tests (Unit Tests), individual component tests (use case based), overall system - multiple component tests (use case based integration/acceptance testing).

#### VII - Model and Construct Other Components As Required

19 - GUI/User/External Interface Components - Optional - Create the GUI component (windows, menus, dialog boxes, panels).

20 - **Persistent Storage Components** - Optional - Create the persistent storage component - data storage tables/stored procedures/triggers.

# UML CASE Study - BankApp with Rational Rose

## 0 - Preliminary - Business Enterprise Models

Business Vision, Objectives, and Organization - Provided Separately

# Business Events (UML Signal Events -Named Stimulus Form or Document) and Processes (UML Use Cases)

Process	Actors	Events/I	Transfor	Events/	Constraints	Description	Reference	Point of
Name		nputs	mation	Output				Contact
Withdraw	Customer,	Withdra	Update	Withdra				
FromAcc	Teller,	wReque	Account	wRecor				
ount	BankDB	st		d				



#### Business Actors, Business Workers, and Business Entities (Problem Domain Entities)

Business Actor (UML Actor)	Business Worker (UML Actor)	<b>Business Entity (UML Class)</b>
Customer	Teller	Account, SavingsAccount,
		CheckingAccount



#### **Business Rules Catalog**

Rule Identifier	Actor, Entity, Process	Description:If	Areas	Reference	Point of
		ConditionalThen Action			Contact
ValidAccount	Account	If AccountNum is Valid			
		then Account is Valid			

#### **Business Interfaces (Named Set of Operations) - Provided by Architect**



Business Patterns Catalog - See Business Modeling with UML by Eriksson and Penker

Business Glossary - to be completed

#### Business Systems Architecture - Provided by Architect



### **I** - Requirements Models

**1 - Requirements:** The BankApp shall manage checking and savings account deposits and withdraws. Future: inquiries, transfers, overdraft, etc. **Inputs/Outputs** TellerGUI to BankApp: sAcctNum, nDeposit, nWithdraw, sText. Inputs/Outputs BankApp to BankDB: sAcctNum, nDeposit, nWithdraw.

**1 - Requirements Traceability Table**: Requirement Number, Name, Reference, Use Case Name, UML Element, Test Case, Description, Responsibility.

Require ment Number	Requirement Name	Reference	Use Case Name	UML Element	Test Case	Description	Responsibility
1.1	DepositToSavingsAc count		DepositToSaving sAccount	BankPkg			
1.2	DepositToCheckingA ccount		DepositToCheckin gAccount	BankPkg			
1.3	WithdrawFromSaving Account		WithdrawFromSa vingAccount	BankPkg			

#### **Requirements Traceability Table (Partial)**

1.4	WithdrawFromChecki	WithdrawFromCh	BankPkg		
	ngAccount	eckingAccount	9		

#### 2 - Requirements - Use Case Diagram - All Use Cases



**Rose Use Case Diagram**: In Browser Window select Use Case View; Rename Main to be Use Case Diagram; Place actors, use cases, and relationships (Rose Unidirectional Association and Generalization) on the diagram; Select each actor - right mouse to enter actor operations; Select Tools - Check Model; Select File - Save.

#### 2 - Requirements - High Level Collaboration Diagram (Context Diagram)



**Rose High Level Collaboration Diagram**: In Browser Window select Use Case View; Select Browse -Interaction Diagram - Use Case View - <New>; Select Collaboration Diagram; Enter Diagram Name; Place objects representing actors on the diagram; Double-click each object then select the actor name from the pull-down list; Place one object in the center of the diagram to represent the system; Double-click the object and enter the system name; Select Rose Object Link symbol and drag between actors and the system ; Select the Rose Text Box "ABC" and enter names of passed objects/data; Select Tools - Check Model; Select File - Save.

#### 2 - Requirements - Use Case Increments:

Increment 1: Checking and Saving Account Deposits and Withdraws Increment 2: Inquiries and Transfers Increment 3: Overdrafts Iterations within each Increment: optimistic, normal, pessimistic

**3 - Requirements Use Case Specification**: Name, Trigger, Input Parameters, Output Return, Precondition/Exception Raised, Postcondition/Exception Raised, Basic/Optimistic Scenario, Alternative/Pessimistic

Scenarios, Business Rules, Test Cases

#### Use Case Specification for WithdrawFromCheckingAccount Use Case

Use Case Name: WithdrawFromCheckingAccount **Trigger**: WithdrawFromCheckingAccount **Input Parameters**: sAcctNum, nWithdraw **Output Return**: sText **Precondition**: ValidAccount = true and nWithdraw <= nCurrentBalance Precondition Exception Raised: To be determined **Description/Transformation**: nCurrentBalance = nCurrentBalance - nWithdraw **Postcondition**: nCurrentBalance < nOldBalance Postcondition Exception: None Related Use Cases: Generalization, Includes, Extends/Extension Point: None Basic Scenario/Optimistic Scenario: Text - to be determined; Diagram - see WithdrawFromCheckingAccount - Optimistic Scenario Sequence Diagram Alternative Scenarios/Pessimistic Scenario: Text - to be determined - Diagram - see WithdrawFromCheckingAccount Activity Diagram Business Rules: ValidAccountRule, AdequateBalanceRule Test Cases: 1 - Optimistic:Inputs: sAcctNum - BGates001, nWithdraw - 100, nCurrentBalance - 1000 Conditions: None, Output: "BGates001 withdraw \$100 OK and recorded", 2 ... To be determined Input and Output Forms: See below

#### Input/Output Forms for WithdrawFromCheckingAccount Use Case:

#### Withdraw Request Form

Customer Account Number \_\_\_\_\_\_ Withdraw Amount

Button-Submit Button-Clear

#### Withdraw Response Form

Customer Account Number \_\_\_\_\_ Withdraw Amount \_\_\_\_\_ Status

Button-OK

3 - Requirements - Sequence Diagram WithdrawFromCheckingAccount - Optimistic Scenario



Note: getApp required to get the top level application for the first transaction

**Rose Sequence Diagram**: In Browser Window select Use Case View; Select the Use Case Diagram; Select a use case; Select Browse - Interaction Diagram - Use Case View - <New>; Select Sequence Diagram; Enter Diagram Name, e.g. WithdrawFromCheckingAccount-OptimisticScenario; Place objects representing actors on the diagram; Double-click each object then select the actor name from the pull-down list; Place one object in the center of the diagram to represent the system; Double-click the object and enter the system name from the pull-down list; Select Rose Object Message symbol and drag between actors and the system; Select Tools - Check Model; Select File - Save.



#### 4 - Requirements Activity Diagram for WithdrawFromCheckingAccount Use Case - All Scenarios

Note: getApp required to get the top level application for the first transaction

**Rose State - Activity Diagram**: In Browser Window select Use Case View; Select the Use Case Diagram to display the diagram; Select a use case; Select Browse - State Diagram; If "State Diagram" is grayed out, then go back to the use case diagram and re-select a use case; Place activity states on the diagram; Place transitions on the diagram by dragging between states; Select Tools - Check Model; Select File - Save.

**5**-Requirements - Product Capabilities: High Reliability, 10 concurrent users, 2 second response time.

**5 - Requirements/Analysis - High Order Concept Model**: External Actors: TellerGUI, BankDB Internal Entities: BankApp, Account, CheckingAccount, SavingsAccount



**Rose High Level Concept Model Diagram**: Recommend do the HOCM with pencil and paper. Alternatively, create a Rose Class Diagram without attributes and operations.

## II - Analysis Model - Rough Sketch

#### 6 - Analysis Class Diagram - Simplest Structure



Rose Class Diagram: See Using Rational Rose

#### 6 - Analysis Package Diagram



**Rose Package Diagram:** In Browser Window select Logical View; Select Browse - Class Diagram - Logical View - <New>. Enter the Package Diagram Name; Place packages on the diagram; To place a dependency relationship, select the dependency arrow from the Toolbar then drag from the source package to the destination package; In the Browser drag each class to the appropriate package; Select Tools - Check Model; Select File - Save.

#### 7 - Analysis Sequence Diagram for WithdrawFromCheckingAccount - Optimistic Scenario



Note: getApp required to get the top level application for the first transaction

Rose Sequence Diagram: See Using Rational Rose

#### 8 - Analysis State Diagram for Account Class



TerminationState

**Rose State - Activity Diagram**: In Browser Window select Logical View; Select the Class Diagram to display the diagram; Select a class; Select Browse - State Diagram; If "State Diagram" is grayed out, then go back to the class diagram and re-select a class; Place states on the diagram; Place transitions on the diagram by dragging between states; Select Tools - Check Model; Select File - Save.

**9 - Analysis - Complex Operations**: To be determined - Activity Diagram and/or Operation Specification for each operation: name, inputs, precondition/exception, transformation, postcondition/exception, business rules, description

**Rose Specifications:** Display the class diagram; Select a class; Press the Right Mouse Button to display the Specification Dialog Box; Select a tab, e.g. Operations Tab; Double-click an operation; Fill-in the operation information. Select Tools - Check Model; Select File - Save.

### III - Design Models - Basis for Coding

**10 - Design Processing Environment:** Linux Version 6.2, GNU C++ Version 6.2, C++ Standard Library, CORBA 3.

### **10 - Design Potential Patterns:**

Enterprise: Distributed CORBA Based, components with public interfaces System (Component to Component): Layered, Session - Entity, Callbacks, Publisher - Subscriber Component: Application - Document, Controller - Entity - Boundary, Facade Class Design: UML, Factory, Transaction Java Language: Java Bean, Enterprise Java Bean, Servlet, RMI

#### 11 - Design Package Diagram



11 - Design Class Diagram Showing Types and Parameters - Goal is completeness for coding



Note: CheckingAccount and SavingsAccount will provide implementation of the polymorphic operations.

#### 11 - Design Sequence Diagram WithdrawFromCheckingAccount - Optimistic Scenario



Note: getApp required to get the top level application for the first transaction

#### 11 - Design Collaboration Diagram WithdrawFromCheckingAccount - Optimistic Scenario

In Rational Rose open sequence diagram and press F5 to automatically create collaboration diagram.



#### 11 - Design Operation Specification for withdraw() in CheckingAccount Class

Use Case Name: withdraw Trigger:withdraw Input Parameters: nWithdraw : int Output Return: boolean Precondition: nWithdraw <= nCurrentBalance Precondition Exception Raised: exInsuffientFunds Description/Transformation: nCurrentBalance = nCurrentBalance - nWithdraw Postcondition: nCurrentBalance < priorCurrentBalance Postcondition Exception: exIncorrectBalance **Basic Scenario/Optimistic Scenario**: See withdrawFromCheckingAccount Sequence Diagram **Alternative Scenarios/Pessimistic Scenario**: See withdrawFromCheckingAccount Activity Diagram **Business Rules:** ValidAccountRule, AdequateBalanceRule

**11 - Design Exception Classes:** Exception Superclass with Exception(),Exception(string); Exception Subclasses: exInsufficientFunds with exInsufficientFunds() and exInsufficientFunds(string); exIncorrectBalance with exIncorrectBalance() and exIncorrectBalance(string).

#### 11 - Design State Diagram for Checking Account Class



TerminationState

### **IV - Implementation Models**

### 12 - Design Processing Environment: UNIX, C++, CORBA

### 13 - Implementation Component Diagram



Implementation Files: TellerGUI.exe, BankApp.exe, BankDB.exe

Component Interface Alternatives: 1) BankApp has single interface IBankApp with all operations exposed

- 2) BankApp exposes IBankApp, ICheckingAccount, & ISavingAccount Interfaces
- 3) BankApp exposes IBankApp, IWithdraw, IDeposit, ICheckingAccount, & ISavingAccount Interfaces

CORBA IDL/C++ needed to describe interfaces

**Rose Component Diagram**: In Browser Window select Component View; Rename Main to be Component Diagram; Double-click the diagram name to display the diagram; Place component and dependency relationships (drag from the client component to the supplier component) on the diagram; In the Browser drag each class to the approprate component; Select Tools - Check Model; Select File - Save.

#### 14 - Implementation Deployment Diagram



**Rose Deployment Diagram**: In Browser Window select Deployment View; Double-click to display the diagram; Place nodes and connection relationships on the diagram; Select Tools - Check Model; Select File - Save.

```
//Interface for BankApp System Using CORBA
```

```
IBankApp
   (from BankPkg)
    + getApp() : IBankApp
    + depositToCheckingAccount(sAcctNum : string, nDeposit : int) : boolean
    + depositToSavingsAccount(sAcctNum : string, nDeposit : int) : boolean
    + withdrawFromCheckingAccount(sAcctNum : string, nWithdraw : int) : boolean
    + withdrawFromSavingsAccount(sAcctNum : string, nWithdraw : int) : boolean
//Sample IDL Interface Code
module BankApp {
interface IBankApp {
  exception exInsuffientFunds;
  IBankApp getApp();
  boolean depositToCheckingAccount (in string sAcctNum, in int nDeposit) ;
  boolean depositToSavingAccount (in string sAcctNum, in int nDeposit)
  boolean withdrawFromCheckingAccount (in string sAcctNum, in int nWithdraw)raises
(exInsuffientFunds);
  boolean withdrawFromSavingAccount (in string sAcctNum, in int nWithdraw) raises
(exInsuffientFunds);
};};
//Sample Java Interface Code Using Remote Method Invocation
import java.rmi.*;
package BankApp;
```

```
public interface IBankApp extends java.rmi.Remote {
  boolean depositToCheckingAccount (String sAcctNum, int nDeposit) throws java.rmi.RemoteException ;
  boolean depositToSavingAccount (String sAcctNum, int nDeposit) throws java.rmi.RemoteException ;
  boolean withdrawFromCheckingAccount (String sAcctNum, int nWithdraw) throws
java.rmi.RemoteException;
  boolean withdrawFromSavingAccount (String sAcctNum, int nWithdraw) throws
java.rmi.RemoteException;
}
//Sample Microsoft IDL Interface Code for COM - Simplified
library BankAppLib {
dispinterface IBankApp {
    IBankApp getApp();
    boolean depositToCheckingAccount (BSTR sAcctNum, int nDeposit) ;
    boolean depositToSavingAccount (BSTR sAcctNum, int nDeposit) ;
    boolean withdrawFromCheckingAccount (BSTR sAcctNum, int nWithdraw);
    boolean withdrawFromSavingAccount (BSTR sAcctNum, int nWithdraw);
};
coclass BankApp {
    dispinterface IBankApp;
}; };
//Sample SOAP (Simple Object Access Protocol) SDL (Service Description Language) with XML -
Incomplete
<?xml version='1.0'?>
<serviceDescription name='BankApp'</pre>
    xmlns='urn:schemas-xmlsoap-org:sdl.2000-01-25'
    xmlns:dt='http://www.w3.org/1999/XMLSchema'
    xmlns:IBankApp='IBankApp'>
<import namespace='IBankApp' location='#IBankApp'/>
    <soap xmlns='urn:schemas-xmlsoap-org:soap-sdl-2000-01-25'>
        <interface name='IBankApp'>
            <requestResponse name='WithdrawFromCheckingAccount'>
                <request ref='IBankApp:WithdrawFromCheckingAccount'/>
                <response ref='IBankApp:WithdrawFromCheckingAccountResponse'/>
            </requestResponse>
        </interface>
        <service>
            <addresses>
                <address uri='http://myserver/IBankApp.asp'/>
            </addresses>
            <implements name='IBankApp'/>
        </service>
    </soap>
    <IBankApp:schema id='IBankApp' targetNamespace='IBankApp'</pre>
xmlns='http://www.w3.org/1999/XMLSchema'>
        <element name='WithdrawFromCheckingAccount'>
        </element>
        <element name='WithdrawFromCheckingAccountResponse'>
            <type>
                <element name='return' type='dt:boolean'/>
            </type>
        </element>
    </IBankApp:schema>
</serviceDescription>
```

### V - Construction

Coding/Naming Standards; Interface Code - CORBA IDL; C++ Code; CASE Tool Scripts/VBA for custom reports/code generation

**Rose Code Generation** - Requires Rose Professional or Enterprise Version. See Help Topic Code Generation. Basic Steps: 1 - Check Model 2 - Create Components 3 - Map/assign classes to components 4 - Set Code Generation Properties 5 - Select a class, component, or package 6 - Generate code 7 - Examine generated code.

**Rose Reverse Engineering** - Requires Rose Professional or Enterprise Version. See Help Topic Reverse Engineering. Use the Model Update Tool.

### **VI - Testing**

Total System/Integration Testing - All Components/Subsystems; Component Testing - Each Component Unit Testing - Each Class. **Test Case Specification:** 

Test Use Case Name: Use Case Name: Use Case Scenario Name: Trigger: Input Parameters: Output Return: Precondition: Precondition Exception Raised: Description/Transformation: Postcondition: Postcondition Exception: Comments:

## **VII - Model and Construct Other Components**

TellerGUI Forms; BankDB Tables: tblWithdraw, tblDeposit, tblSavingAccount, tblCheckingAccount

# Summary

This tutorial provided by Richard Felsinger, <u>www.felsinger.com</u>, 960 Scottland Dr, Mt Pleasant, SC 29464 <u>dick@felsinger.com</u> or dfelsinger@home.com 843-881-3648. Comprehensive UML Model Building and Java/C++/VB Development and Training Available.