JavaOne

Sun's 2000 Worldwide Java Developer Conference

Enterprise Application Development Using UML, Java™ Technology and XML

Will Howery
CTO
Passage Software LLC

Introduction

- Effective management and modeling of enterprise applications
- Web and business-to-business applications
- Messaging environments
  - Loosely coupled
  - Asynchronous
  - Fault tolerant
- Java technology provides the descriptive language
- XML provides the data representation
- UML provides the notational language
- Manage complexity for deploying n-tier enterprise applications

Overview

- A process and examples for building UML applications with XML messages through multiple distributed server containers
- Illustrate a complete UML design for n-tier application web development
- Implement a web based logon and user profile application
- Utilizing
  - UML
  - Java technology
  - XML DTD/schema definitions

Tools

- Examples presented in UML using Rational Rose with UML Factory
- The examples will be implemented with JAR components that can be deployed into multiple configurations
- UML Factory will generate, deploy, and animate the examples through the UML diagrams

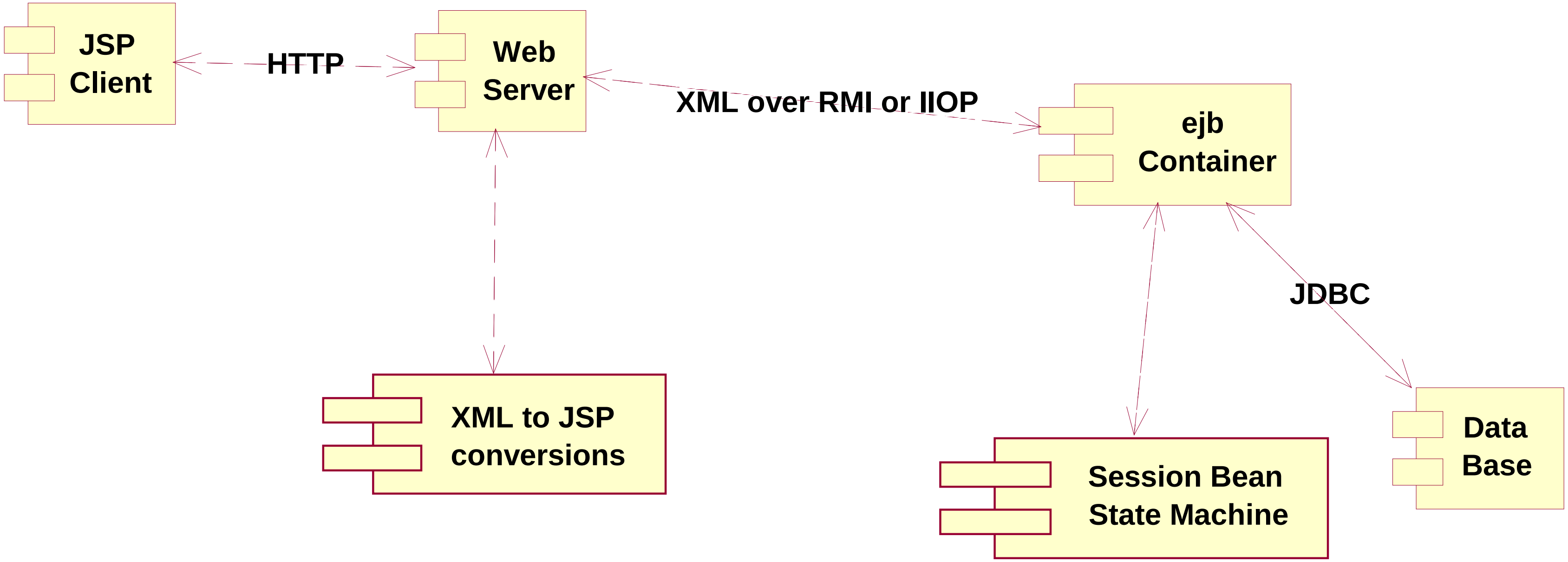
Technologies

- UML:
  - Unified Modeling Language will describe the static and dynamic behavior for applications
- XML:
  - Extensible Markup Language describes document information for building pages and carrying messages through the application tiers
- JSP™ components:
  - JavaServer Pages™ technology-based components are used to define dynamic HTML interface pages

Technologies

- XSL:
  - Extensible Style Sheet language is used to transform the XML documents into dynamic HTML interface pages
- EJB™ specification
  - Enterprise JavaBeans™ specification is used for data access and manipulation
- JMS:
  - Java™ Message Service API provides an asynchronous fault tolerant messaging capability between application tiers

Sample JSP™ Client to J2EE™ Application Container

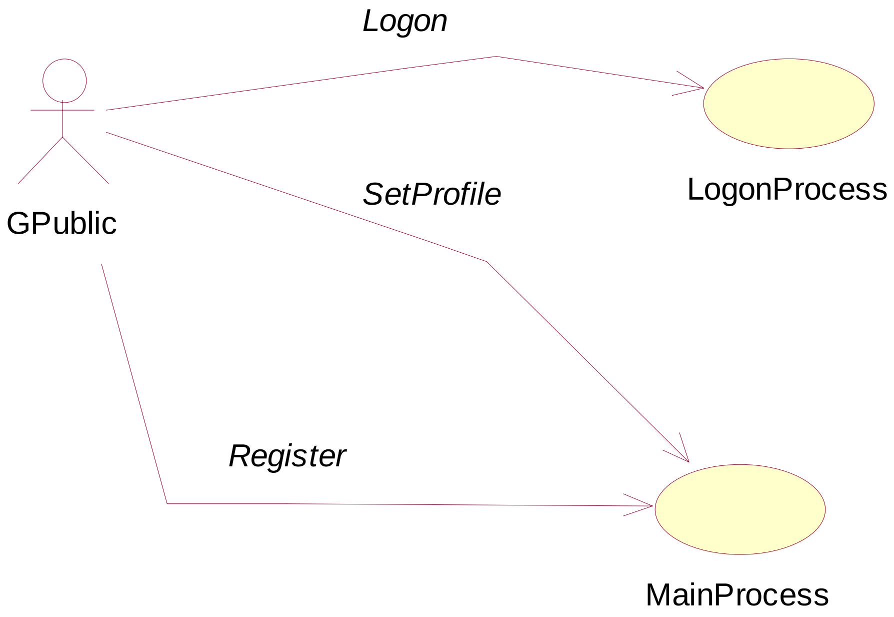


UML Model Overview

- Use Case Diagram
- Collaboration Diagram
- Class Diagrams
- State Diagrams
- Activity Diagrams
- Deployment Diagram

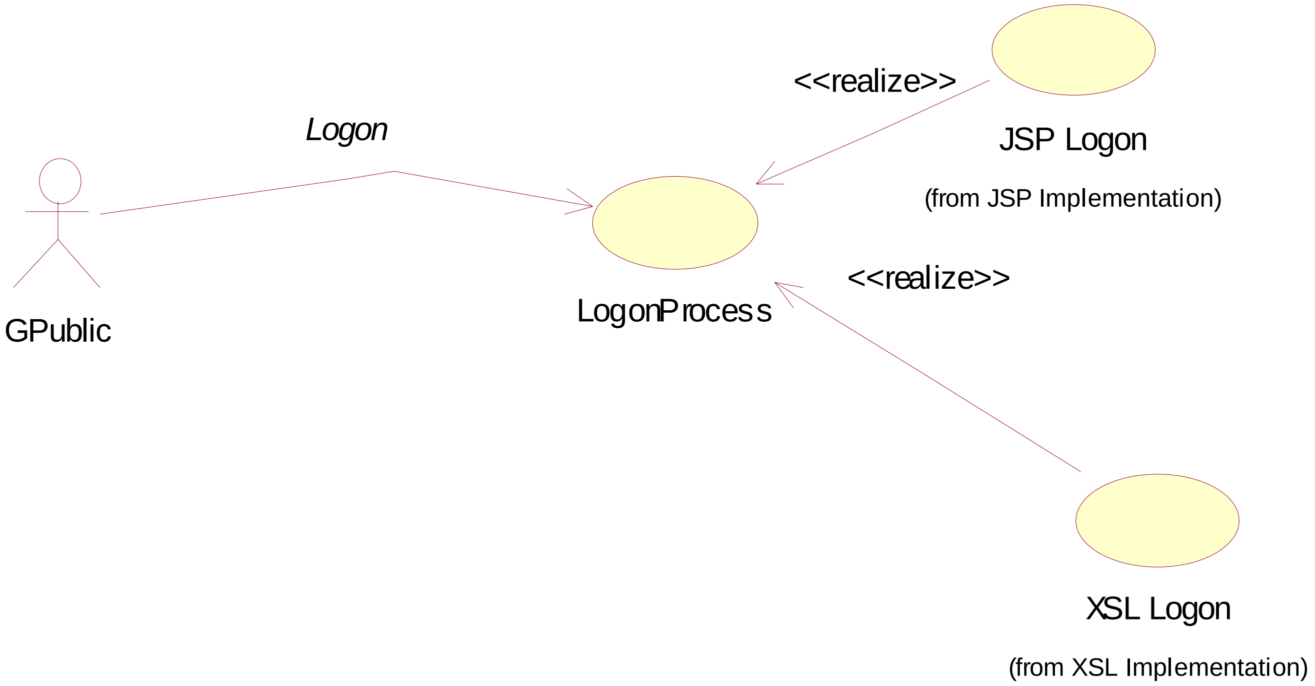
Modeling the Application

- Use Case
  - Use Case Diagrams define the high-level interactions between external actors and system processes. The Use Case diagram must have an actor, an interaction, and a process



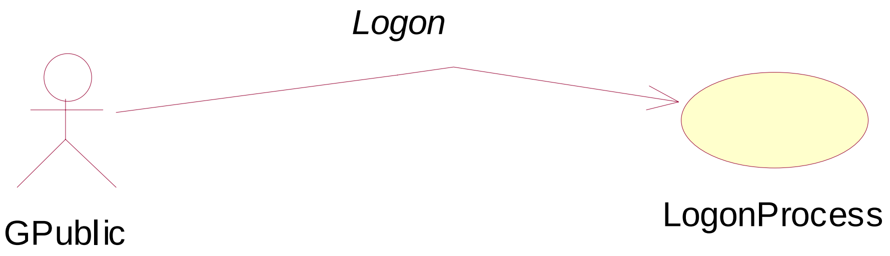
Alternate Implementations

- Realize
  - The “realize” stereotype on a Use Case interaction defines alternate mechanisms for implementing the same process



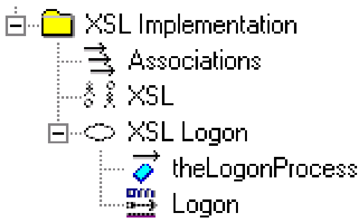
User Artifacts

- Each interaction on a Use Case process contains artifacts, these artifacts are tangible attributes provided to the Actor or input artifacts from the Actor to the process.
For a Logon:
  - UserName
  - UserPassword

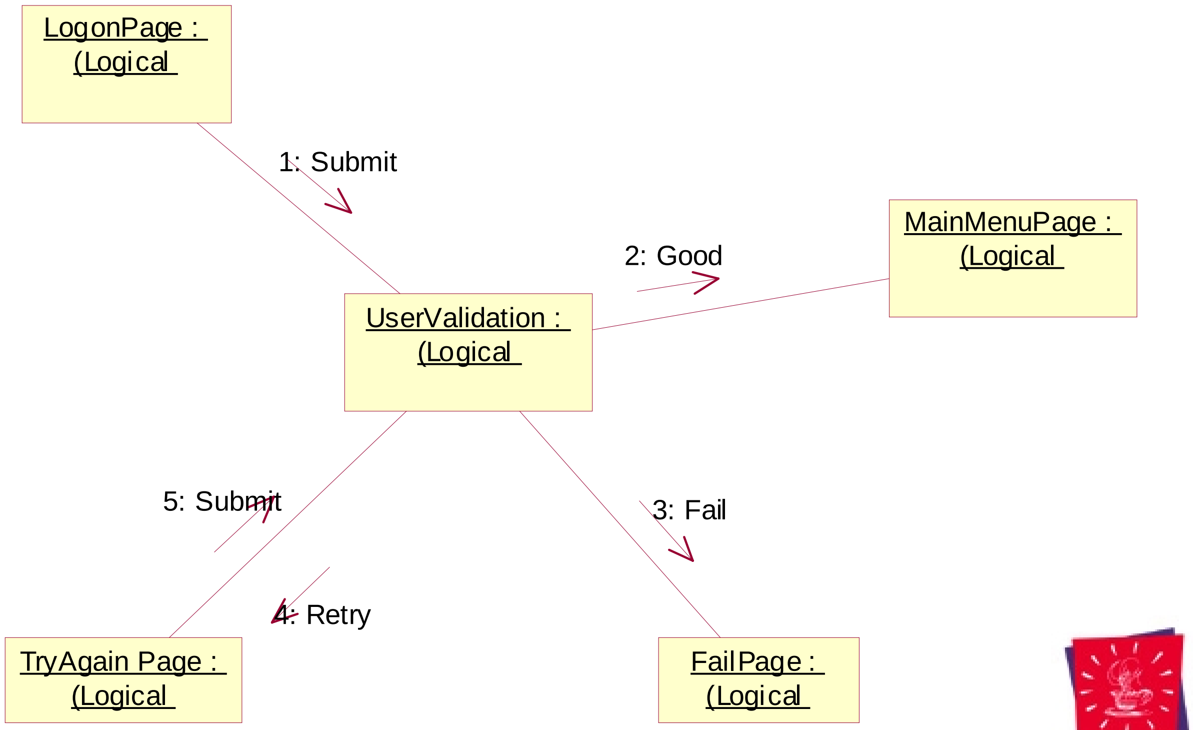


Collaboration Diagram

- Collaboration Diagrams define the implementation for each Use Case interaction

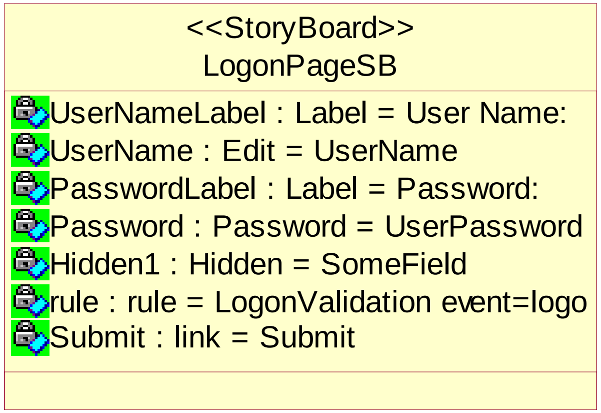


- XSL Implementation
  - Associations
  - XSL
  - XSL Logon
    - theLogonProcess
    - Logon



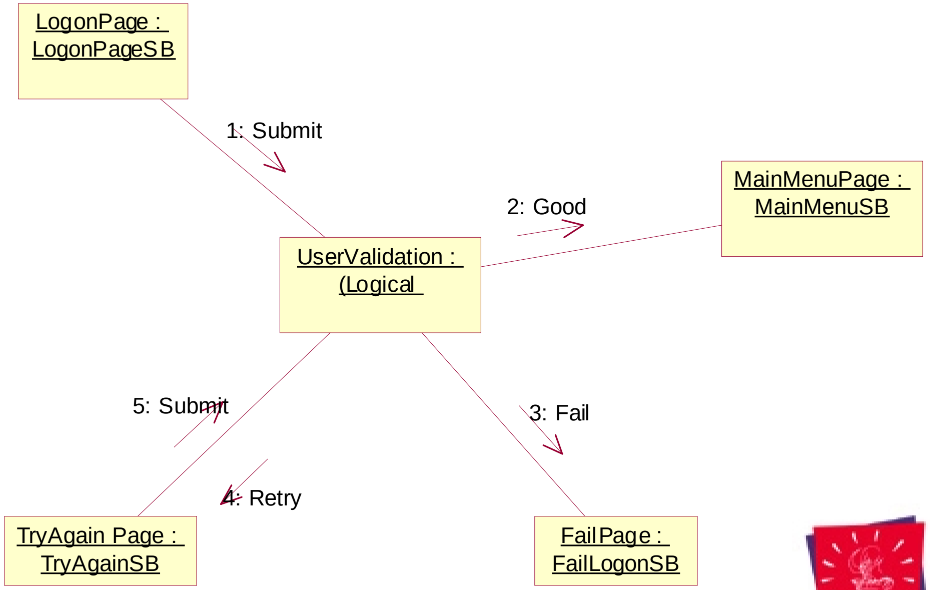
Storyboard Interface Objects

- Storyboard interface objects identify user input and output artifacts. These objects will identify the types of artifacts used within the system for the Use Case interaction



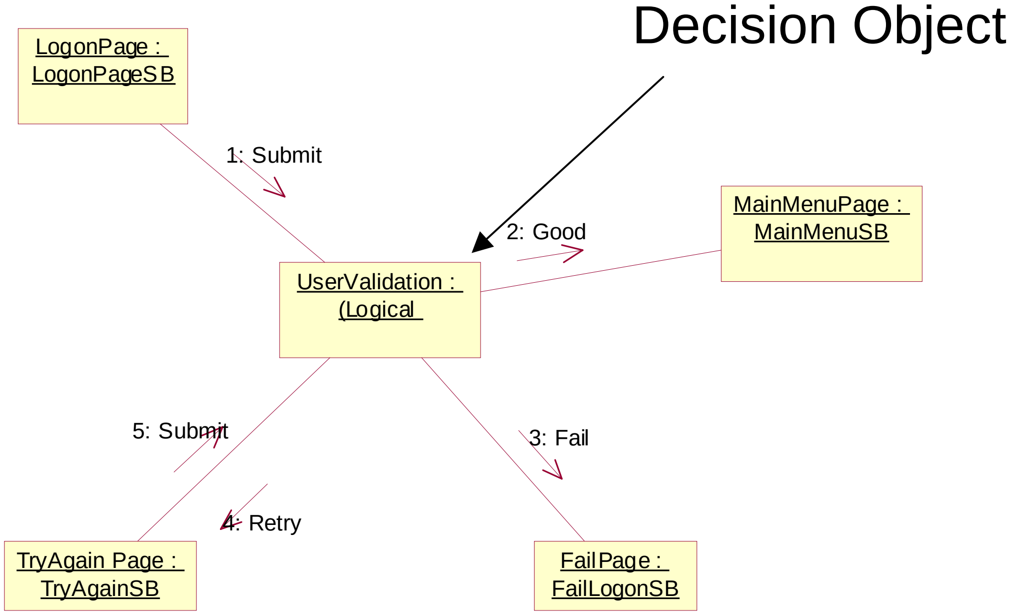
<<StoryBoard>>
LogonPageSB

UserNameLabel : Label = User Name:
UserName : Edit = UserName
PasswordLabel : Label = Password:
Password : Password = UserPassword
Hidden1 : Hidden = SomeField
rule : rule = LogonValidation event=logo
Submit : link = Submit



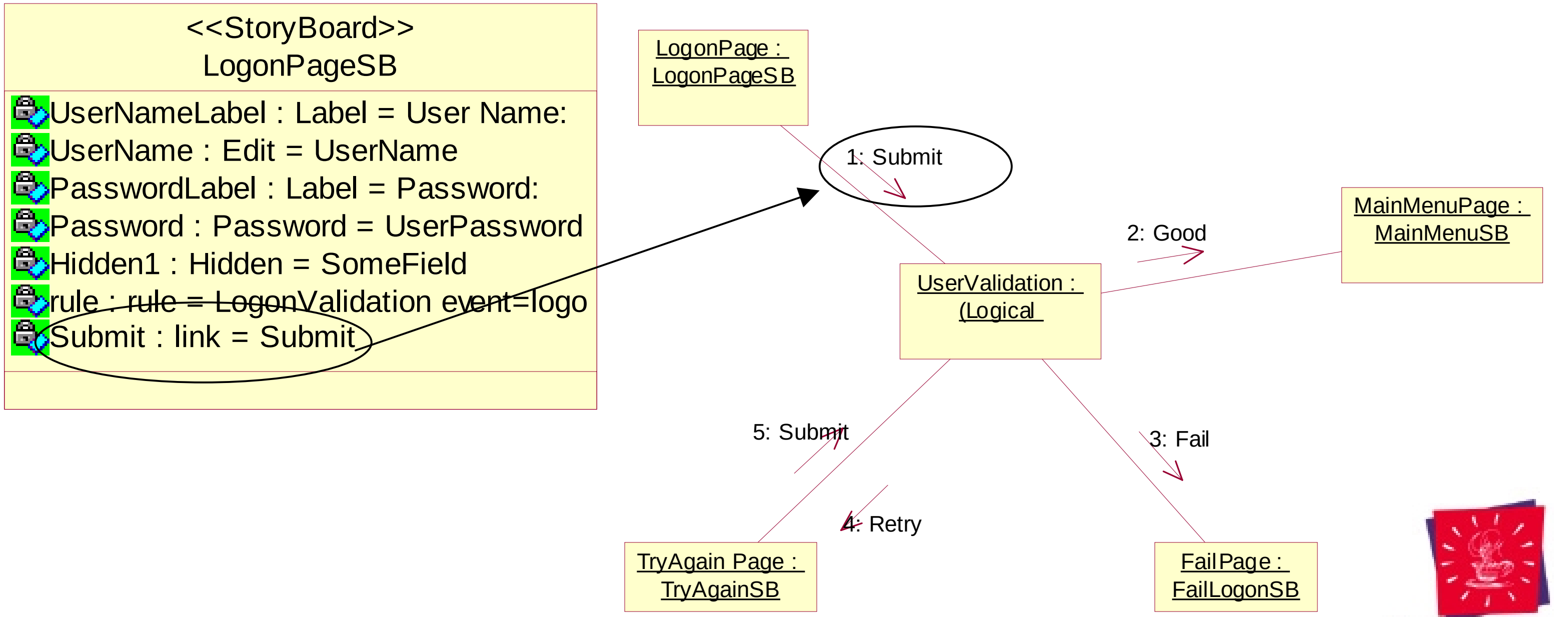
Decision Control Process Flow Objects

- Through the Collaboration diagram choices or decisions are made directing the diagram flow. Control process flow objects define the domain logic class specifications that will govern these decisions



Events

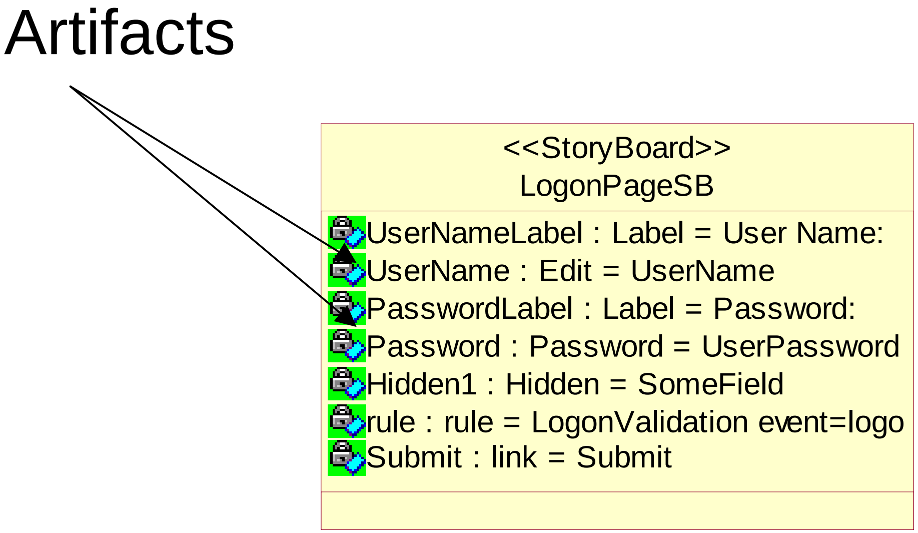
- Collaboration Diagram events show the linking between various collaboration diagram objects



<<StoryBoard>>
LogonPageSB
UserNameLabel : Label = User Name:
UserName : Edit = UserName
PasswordLabel : Label = Password:
Password : Password = UserPassword
Hidden1 : Hidden = SomeField
rule : rule = LogonValidation event=logo
Submit : link = Submit
LogonPage : LogonPageSB
1: Submit
2: Good
MainMenuPage : MainMenuSB
UserValidation : (Logical
5: Submit
3: Fail
4: Retry
TryAgain Page : TryAgainSB
FailPage : FailLogonSB

Storyboard Class Specifications

- The storyboard class specification identifies the ordered set of artifacts and events that this storyboard object element will contain



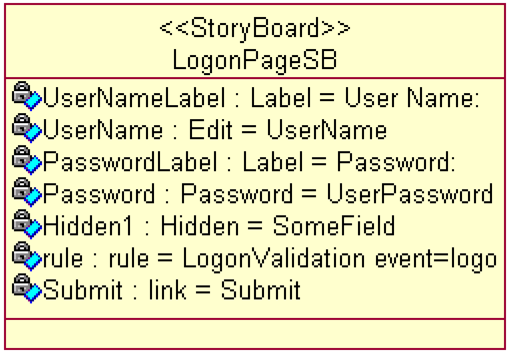
Artifacts

<<StoryBoard>>
LogonPageSB

UserNameLabel : Label = User Name:
UserName : Edit = UserName
PasswordLabel : Label = Password:
Password : Password = UserPassword
Hidden1 : Hidden = SomeField
rule : rule = LogonValidation event=logo
Submit : link = Submit

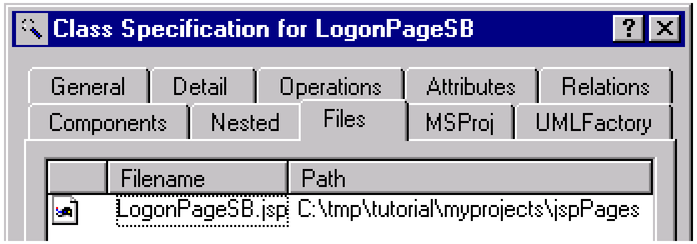
HTML Pages

- Each storyboard class specification is linked to an HTML page that will be processed to create the XSL or JSP interface definition



<<StoryBoard>>
LogonPageSB

UserNameLabel : Label = User Name:
UserName : Edit = UserName
PasswordLabel : Label = Password:
Password : Password = UserPassword
Hidden1 : Hidden = SomeField
rule : rule = LogonValidation event=logo
Submit : link = Submit



Class Specification for LogonPageSB

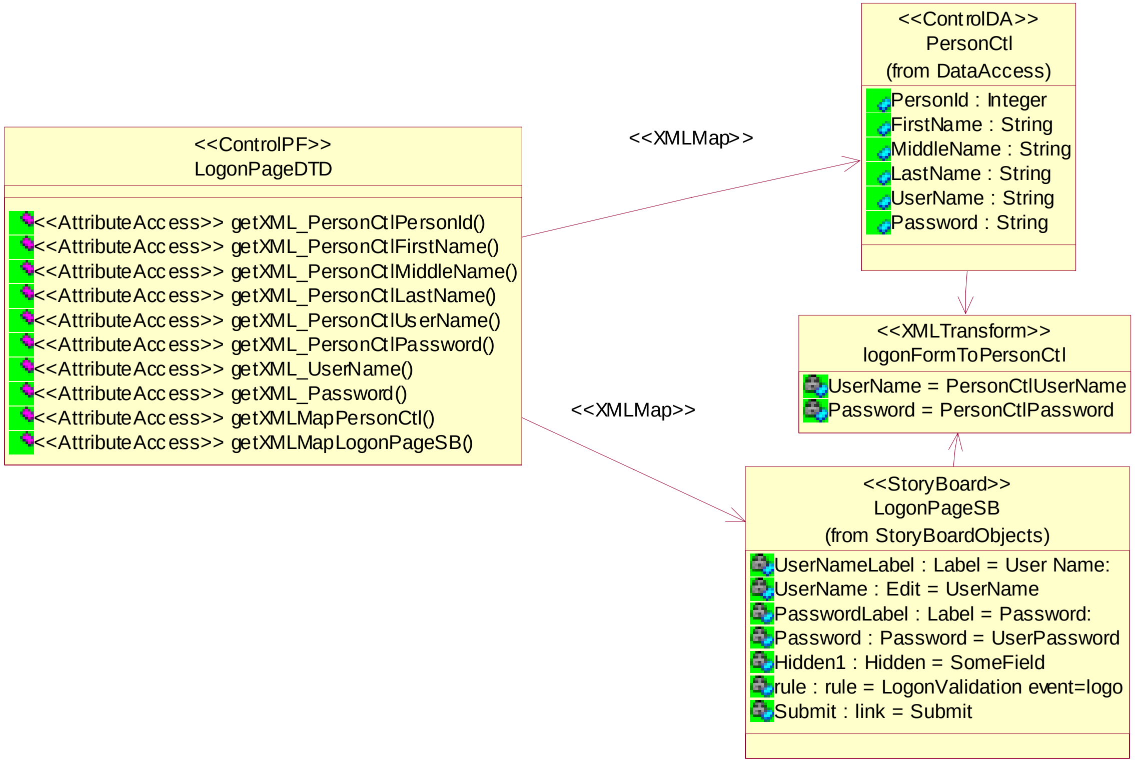
Filename	Path
LogonPageSB.jsp	C:\tmp\tutorial\myprojects\jspPages

HTML Pages

- The HTML pages will contain tokens identifying the XML abstract semantic names representing the use case artifacts coming and going from the control process flow

XML DTD/Schema Modeling

- Modeling the XML schema information within UML provides a visual representation of the XML documents structure. The modeled XML document also provides runtime information and model time checking for collating the use case artifacts with the XML elements
  - Sample Company Person XML relating to the LogonPage



<<ControlPF>>
LogonPageDTD

<<AttributeAccess>> getXML_PersonCtlPersonId()
<<AttributeAccess>> getXML_PersonCtlFirstName()
<<AttributeAccess>> getXML_PersonCtlMiddleName()
<<AttributeAccess>> getXML_PersonCtlLastName()
<<AttributeAccess>> getXML_PersonCtlUserName()
<<AttributeAccess>> getXML_PersonCtlPassword()
<<AttributeAccess>> getXML_UserName()
<<AttributeAccess>> getXML_Password()
<<AttributeAccess>> getXMLMapPersonCtl()
<<AttributeAccess>> getXMLMapLogonPageSB()

<<XMLMap>>

<<ControlDA>>
PersonCtl
(from DataAccess)

PersonId : Integer
FirstName : String
MiddleName : String
LastName : String
UserName : String
Password : String

<<XMLMap>>

<<XMLTransform>>
logonFormToPersonCtl

UserName = PersonCtlUserName
Password = PersonCtlPassword

<<StoryBoard>>
LogonPageSB
(from StoryBoardObjects)

UserNameLabel : Label = User Name:
UserName : Edit = UserName
PasswordLabel : Label = Password:
Password : Password = UserPassword
Hidden1 : Hidden = SomeField
rule : rule = LogonValidation event=logo
Submit : link = Submit

XML Mappings

- UML Factory provides an abstract mapping between XML elements and semantic names. These Mappings allow isolation between the arbitrary physical representation of a data element, and a logical name or handle to access and manipulate that element
  - Semantic Names
    - Semantic Names identify abstract textual identifiers for elements, or attributes, within the XML document
  - XML Element Mappings
    - XML element mappings identify the arbitrary physical XML element definition. The XML mappings identify XML text, cardinality, attributes, and schema information

XML Mapping Example

- Generated semantic mappings to sample Company Person XML

```
mXMLMaptestDTD = new XMLMap();
mXMLMaptestDTD.addMapping("TestXML", "xml");
mXMLMaptestDTD.addMapping("XmlName", "xml/name");
mXMLMaptestDTD.addMapping("Company", "xml /company");
mXMLMaptestDTD.addMapping("CompanyName", "xml/company#name");
mXMLMaptestDTD.addMapping("CompanyCity", "xml/company#city");
mXMLMaptestDTD.addMapping("Person", "xml/company* /person");
mXMLMaptestDTD.addMapping("FirstName", "xml/company/person#fname");
mXMLMaptestDTD.addMapping("LastName", "xml/company/person#lname");
```

XML Mappings to "JSP™ tags"

- The token semantic names within the HTML page associate with storyboard class specifications are substituted with methods to extract semantic data from the XML document representing the Storyboard interface object

XML Mappings to XSL tags

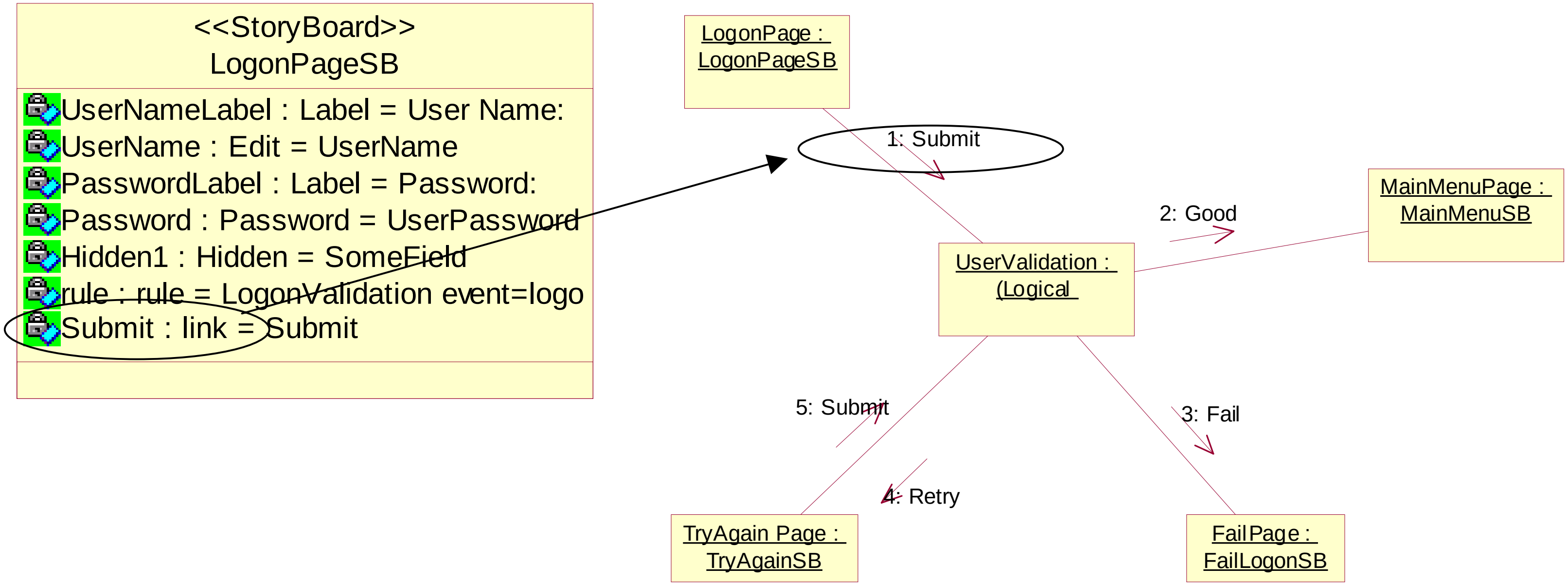
- In like manner, the token semantic names within the HTML page associated with storyboard class specifications are replaced with XSL syntax to transform the XML document representing the Storyboard interface object

Storyboard Events

- The storyboard class specification objects contain events that will fire into the application, transitioning through the designed collaboration diagrams
  - If the target object of a collaboration diagram event is another interface element then that storyboard interface element will be displayed
  - If the target object is a decision point then the class specification defined for that decision logic would be sent the event through the UML state machine implementation

Storyboard Events

- The Logon Page Submit link event



<<StoryBoard>>
LogonPageSB

UserNameLabel : Label = User Name:
UserName : Edit = UserName
PasswordLabel : Label = Password:
Password : Password = UserPassword
Hidden1 : Hidden = SomeField
rule : rule = LogonValidation event=logo
Submit : link = Submit

LogonPage : LogonPageSB

1: Submit

UserValidation : (Logical

2: Good

MainMenuPage : MainMenuSB

3: Fail

FailPage : FailLogonSB

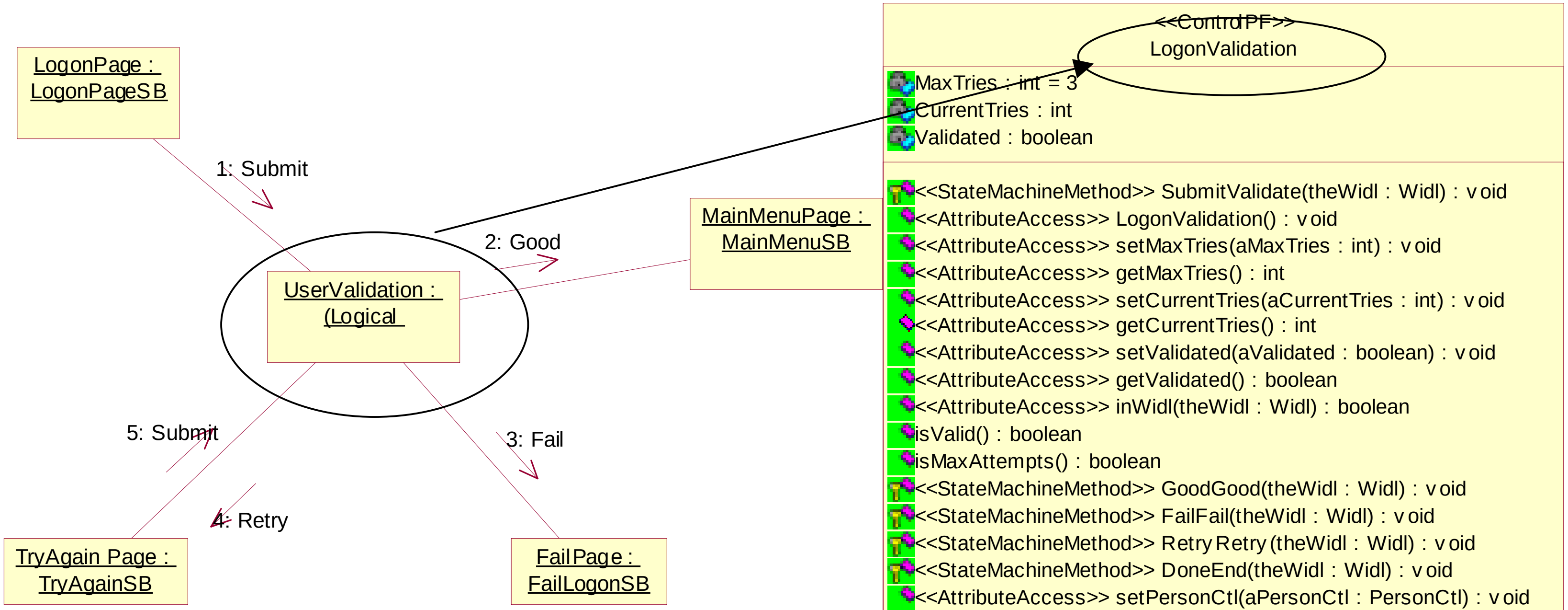
5: Submit

4: Retry

TryAgain Page : TryAgainSB

Collaboration Decisions

- Decisions are implemented by UML logical classes



LogonPage : LogonPageSB

1: Submit

UserValidation : (Logical

2: Good

MainMenuPage : MainMenuSB

3: Fail

FailPage : FailLogonSB

4: Retry

5: Submit

TryAgain Page : TryAgainSB

<<ControlPF>>
LogonValidation

MaxTries : int = 3
CurrentTries : int
Validated : boolean

<<StateMachineMethod>> SubmitValidate(theWidl : Widl) : void
<<AttributeAccess>> LogonValidation() : void
<<AttributeAccess>> setMaxTries(aMaxTries : int) : void
<<AttributeAccess>> getMaxTries() : int
<<AttributeAccess>> setCurrentTries(aCurrentTries : int) : void
<<AttributeAccess>> getCurrentTries() : int
<<AttributeAccess>> setValidated(aValidated : boolean) : void
<<AttributeAccess>> getValidated() : boolean
<<AttributeAccess>> inWidl(theWidl : Widl) : boolean
isValid() : boolean
isMaxAttempts() : boolean
<<StateMachineMethod>> GoodGood(theWidl : Widl) : void
<<StateMachineMethod>> FailFail(theWidl : Widl) : void
<<StateMachineMethod>> RetryRetry(theWidl : Widl) : void
<<StateMachineMethod>> DoneEnd(theWidl : Widl) : void
<<AttributeAccess>> setPersonCtl(aPersonCtl : PersonCtl) : void
<<AttributeAccess>> getPersonCtl() : PersonCtl

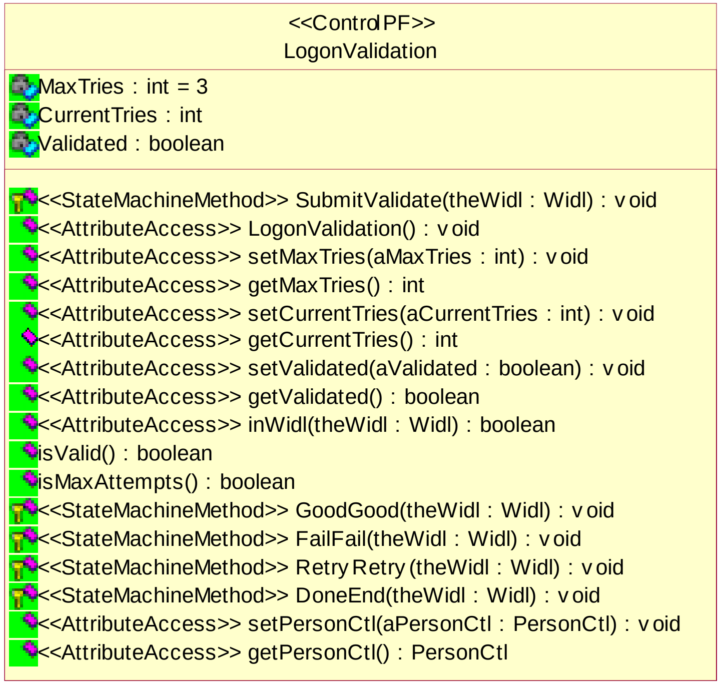
Control Process Flow Class Specification

- The control process flow class specification provides the logical implementation making decisions through the use case collaboration process. The UML control process flow stereotyped class specification is generated into Java™ source code with all the static and dynamic behavior required for:
  - Receiving input events
  - Managing persistent data
  - Manipulating Interface XML documents
  - Returning information to the process
  - Maintaining state information of the application

Receiving Events

- When a user activates an event from an interface object, that event is sent into the state machine. All information coming into the state machine is contained within an XML document

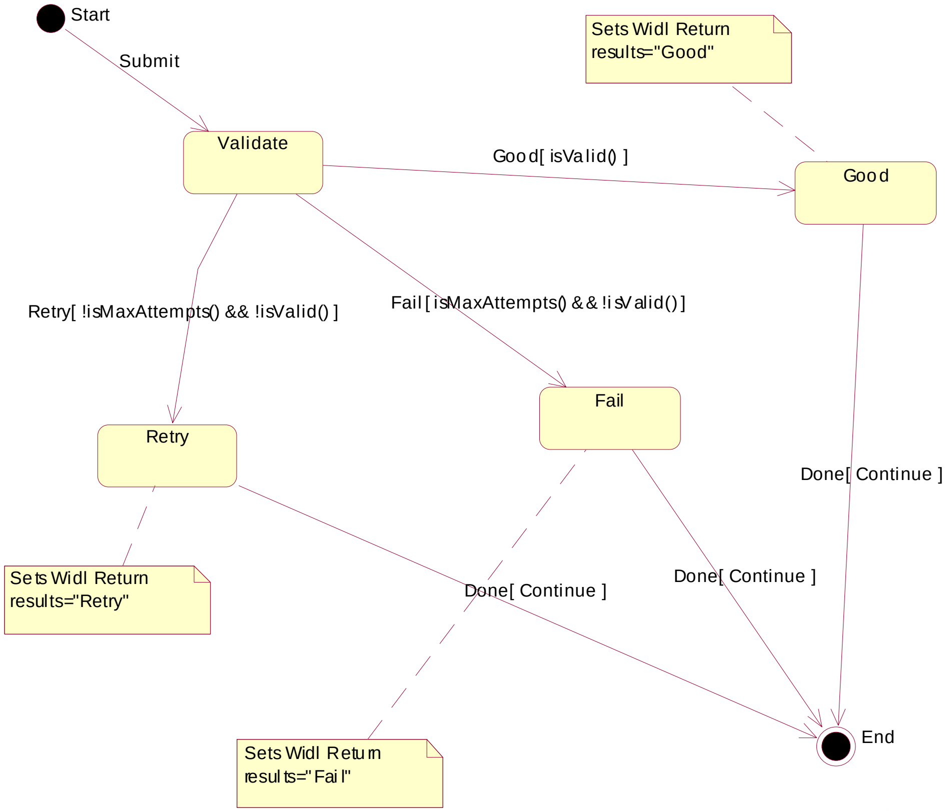
Control Process Class State Machines



<<ControlPF>>
LogonValidation

MaxTries : int = 3
CurrentTries : int
Validated : boolean

<<StateMachineMethod>> SubmitValidate(theWidl : Widl) : void
<<AttributeAccess>> LogonValidation() : void
<<AttributeAccess>> setMaxTries(aMaxTries : int) : void
<<AttributeAccess>> getMaxTries() : int
<<AttributeAccess>> setCurrentTries(aCurrentTries : int) : void
<<AttributeAccess>> getCurrentTries() : int
<<AttributeAccess>> setValidated(aValidated : boolean) : void
<<AttributeAccess>> getValidated() : boolean
<<AttributeAccess>> inWidl(theWidl : Widl) : boolean
isValid() : boolean
isMaxAttempts() : boolean
<<StateMachineMethod>> GoodGood(theWidl : Widl) : void
<<StateMachineMethod>> FailFail(theWidl : Widl) : void
<<StateMachineMethod>> RetryRetry(theWidl : Widl) : void
<<StateMachineMethod>> DoneEnd(theWidl : Widl) : void
<<AttributeAccess>> setPersonCtl(aPersonCtl : PersonCtl) : void
<<AttributeAccess>> getPersonCtl() : PersonCtl



WIDL

- WIDL is a Web Interface Definition Language specification
  - The WIDL contains an event, process, and structured records defining behavior and data together within a single XML document package
  - Event or Method
    - The method described within the WIDL is the event name coming into the state machine
  - Process
    - The WIDL process attribute identifies the executable Java class that will receive the WIDL and respond to the method event. The process name can be a fully qualified Java class or an abstract object name. Containers receiving the WIDL input from either session beans, JMS messaging Queue implementations etc., must have a mechanism to late bind the WIDL event to the correct process

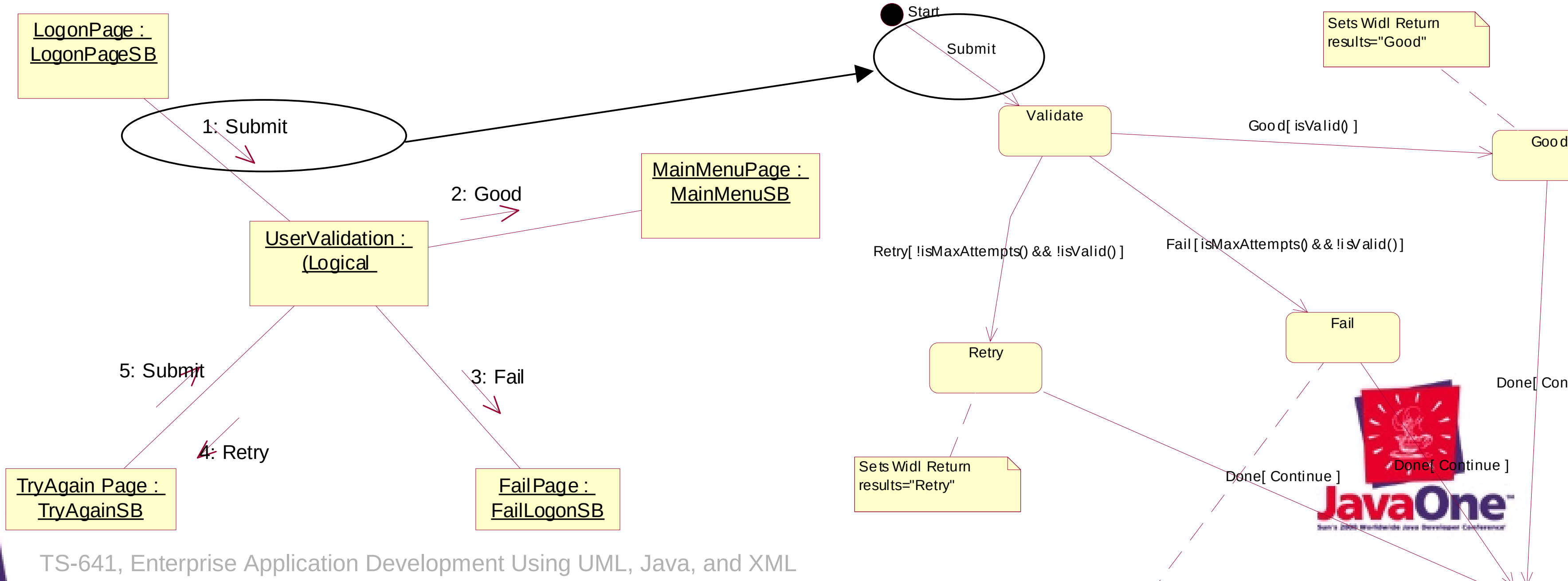
WIDL Records

The WIDL contains three records for input information, output information and return information. These records can be populated with attributes or complete XML documents

- Input Record
  - Our example for the Web application will use the input record to contain information coming from the Web interface into the process logic
- Output Record
  - The example WIDL use the output record of the WIDL to contain Return Page or XML document information back to the Web interface
- Return Record
  - The WIDL return record contains results information for evaluating decisions to the collaboration process flow. The result Attribute within the WIDL Return record contains the return events from the control process flow class specification

State Machines

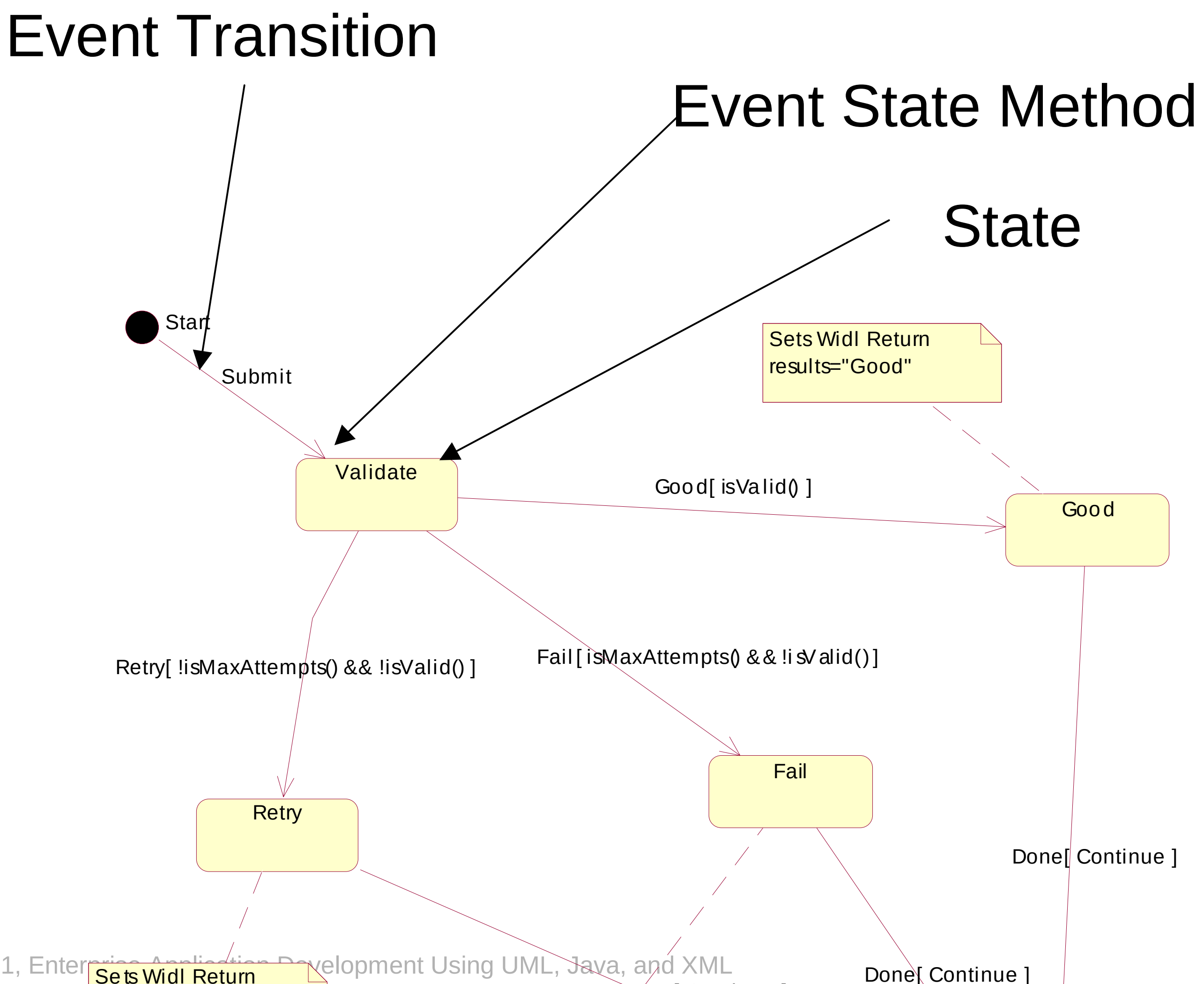
- State machines define the possible events coming into a dynamic class and the state transitions for those events. The state machine diagram provides a readable definition of the dynamic behavior for a given class specification


LogonPage : LogonPageSB
1: Submit
2: Good
MainMenuPage : MainMenuSB
UserValidation : (Logical
5: Submit
3: Fail
4: Retry
TryAgain Page : TryAgainSB
FailPage : FailLogonSB
Start
Submit
Validate
Sets Widl Return results="Good"
Good[isValid()]
Good
Retry[!isMaxAttempts() && !isValid()]
Fail [isMaxAttempts() && !isValid()]
Retry
Fail
Done[Con
Sets Widl Return results="Retry"
Done[Continue]
Done[Continue]

States and Transitions

- States
  - – States define the process stops through the state machine
- Event Transitions
  - – Transitions define the event causing a transition from one state to another state
- Event State Methods
  - – Event state methods are the implied action behaviors when an event transition occurs. These event state methods are implemented through UML activity diagrams

State Diagram Elements



Event Transition

Event State Method

State

Start

Submit

Validate

Sets Widl Return results="Good"

Good[isValid()]

Good

Retry[!isMaxAttempts() && !isValid()]

Fail[isMaxAttempts() && !isValid()]

Retry

Fail

Done[Continue]

Sets Widl Return

Done[Continue]

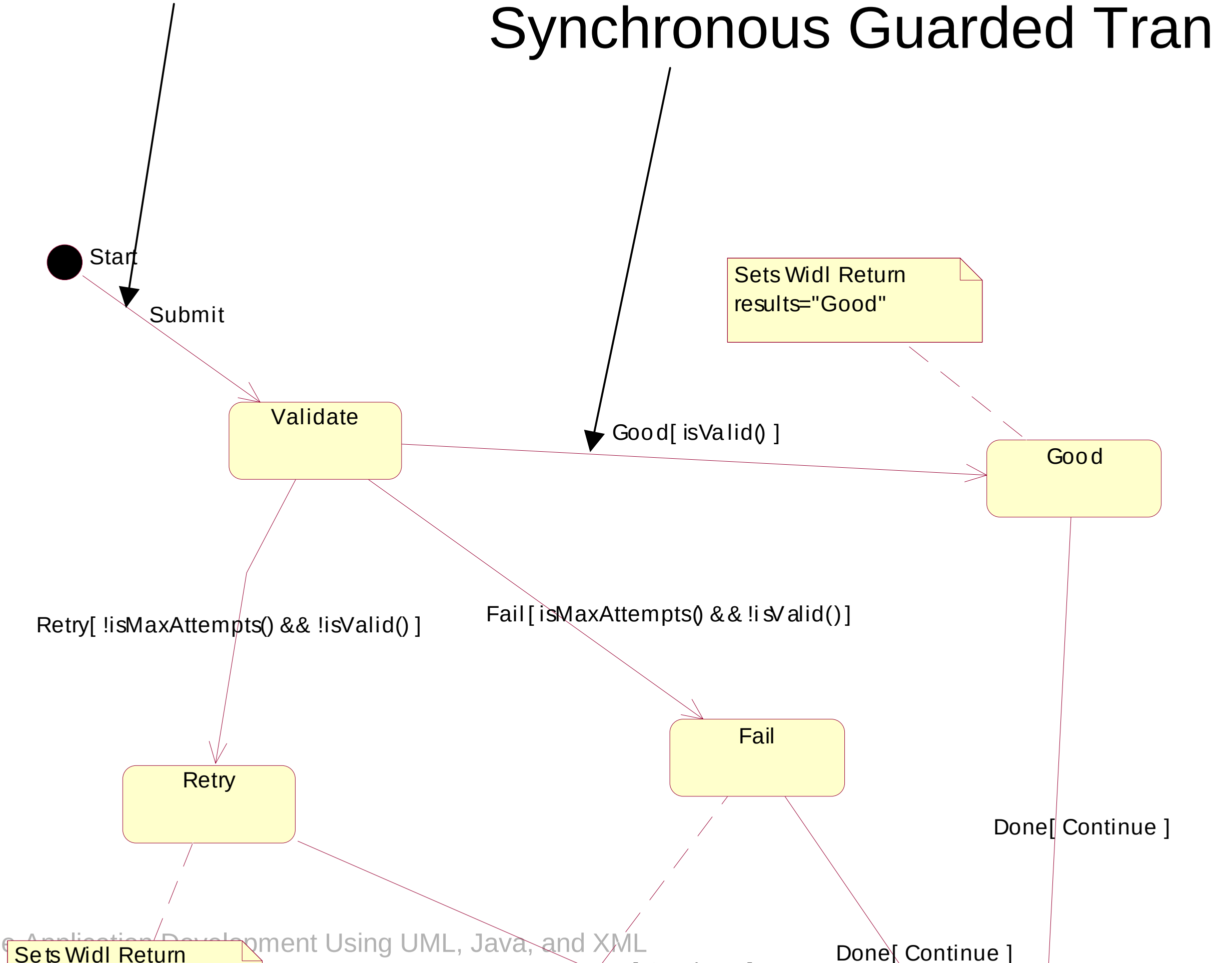
State Timing

- Guard Conditions
  - Guard conditions imply a synchronous transition exiting a given state. The guard conditions contain boolean logic to determine which exit transition will be traversed
- Asynchronous
  - Asynchronous events have no guard conditions and are triggered from some external source. That source is typically a user interface link artifacts
- Synchronous
  - Synchronous events transition automatically and internal to the state machine. Synchronous events are identified as exit events from a state containing guard conditions

Transition Timing Example

Asynchronous Transition

Synchronous Guarded Transition



Start

Submit

Validate

Sets Widl Return
results="Good"

Good[isValid()]

Good

Retry[!isMaxAttempts() && !isValid()]

Fail[isMaxAttempts() && !isValid()]

Retry

Fail

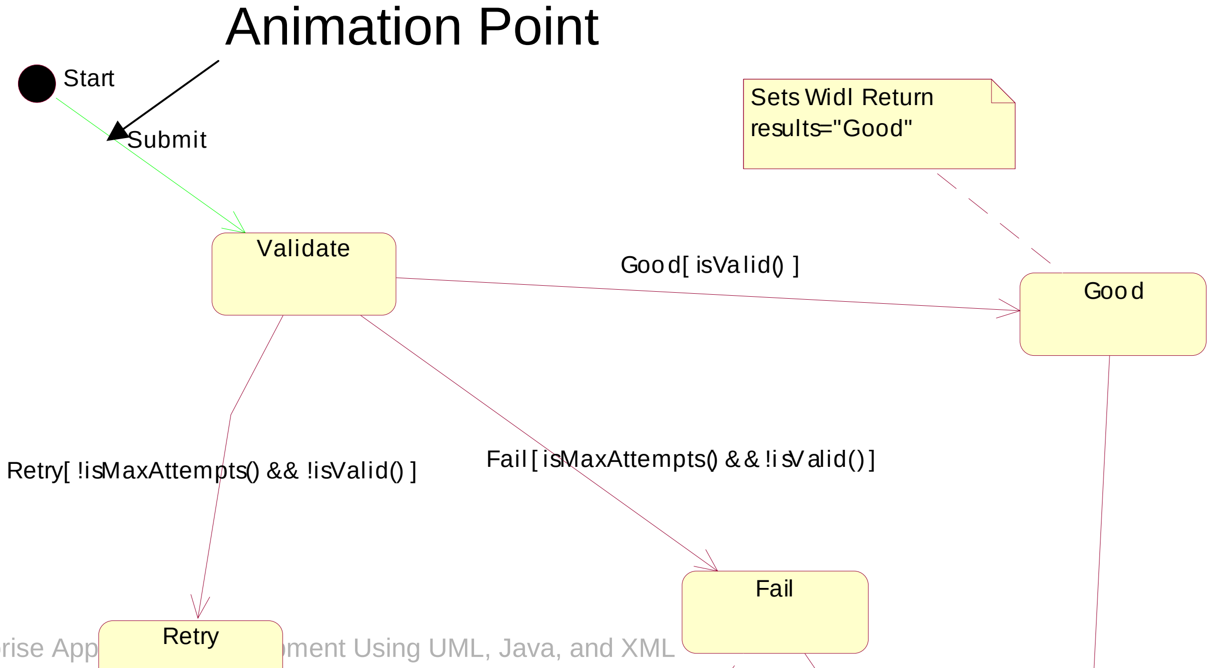
Done[Continue]

Sets Widl Return

Done[Continue]

Setting Animation Points

- Within the state machine any transition can be identified within the UML diagram as an animation point. When the application is executed the UML document will be displayed selecting the animation location

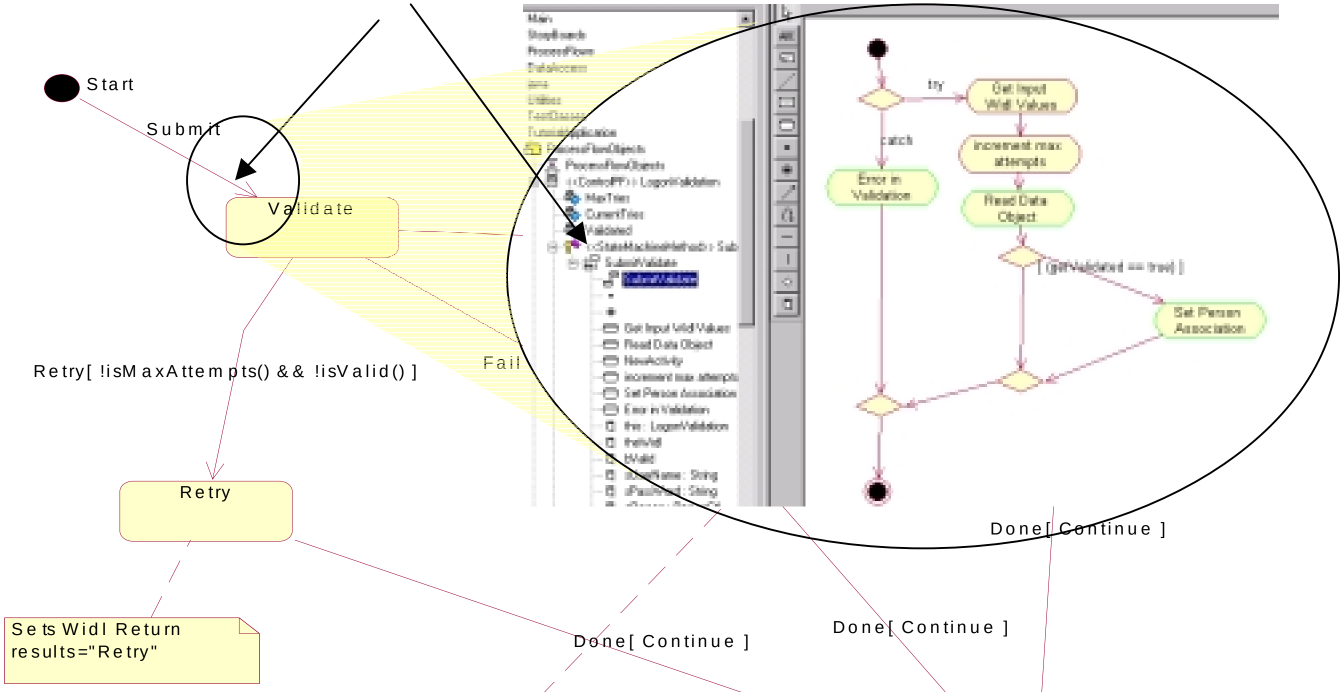

Animation Point

Start
Submit
Sets Widl Return results="Good"
Validate
Good[isValid()]
Good
Retry[!isMaxAttempts() && !isValid()]
Fail [isMaxAttempts() && !isValid()]
Retry
Fail

Activity Diagrams

- Activity Diagrams implement the behavior of algorithms for UML class specification methods

Event State Method

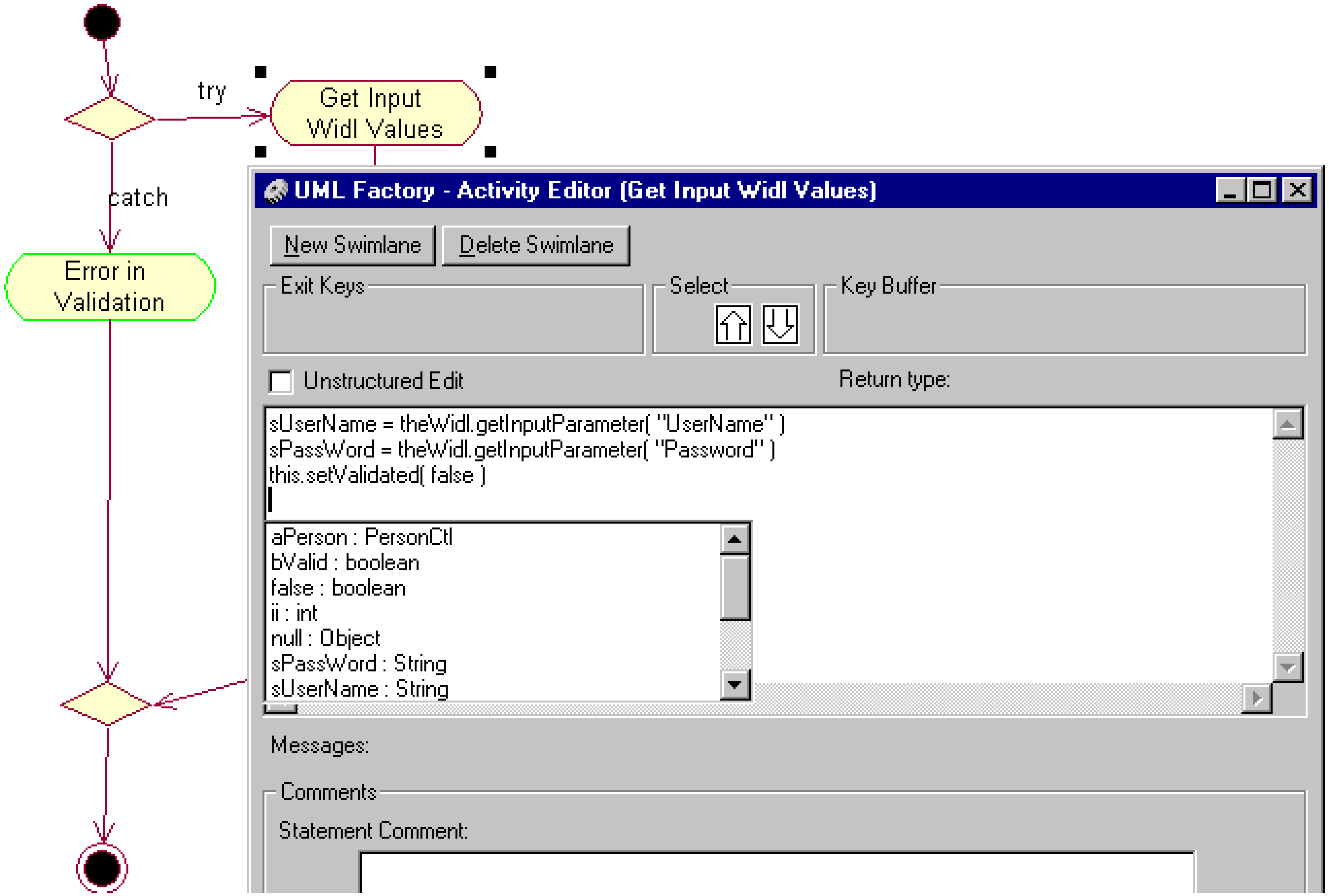


UML Object Navigation Notation

- Activity Diagrams are represented in UML Object Navigation Notation. This notation manipulates the objects with their public attributes and methods. Using the object notation and Activity Editor within UML Factory you can define the complete behavior for a method implementation

```
sUserName = theWidl.getInputParameter( "UserName" )
sPassWord = theWidl.getInputParameter( "Password" )
this.setValidated( false )
```

Designing an Activity



□ Unstructured Edit

```
sUserName = theWidl.getInputParameter( "UserName" )
sPassWord = theWidl.getInputParameter( "Password" )
this.setValidated( false )
```

```
aPerson : PersonCtl
bValid : boolean
false : boolean
ii : int
null : Object
sPassWord : String
sUserName : String
```

Accessing a Data Base

- Through the activity diagram implementations we can manipulate our database objects, defining the operations for all our database elements as required

```
aPerson.PersonCtl(  )
aPerson.setFirstName( sUserName )
aPerson.setPassword( sPassWord )
this.setValidated( aPerson.load(  ) )
```

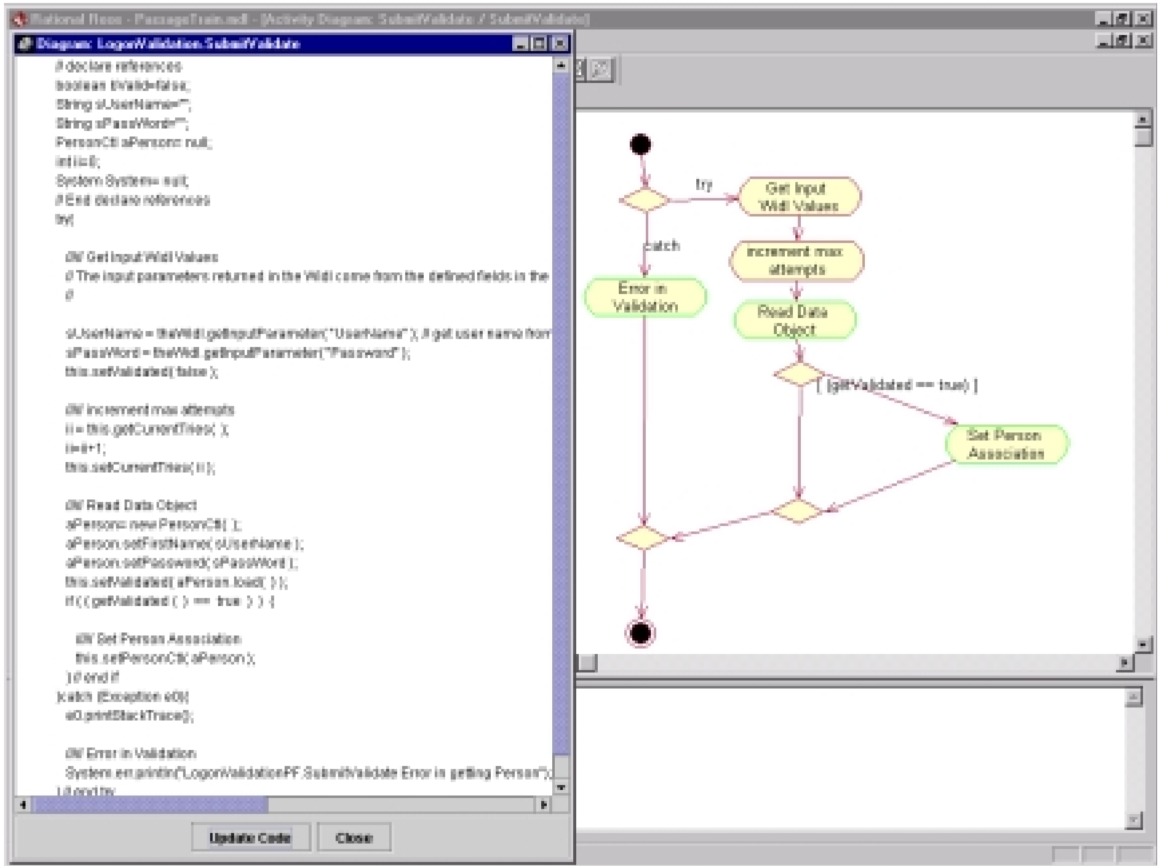
Manipulating XML

- Within the activity diagrams we also define the XML manipulation for retrieving and building the XML documents going between the process flow logic and user interface elements

```
sValue= new String( "PassageSoftware" );
    aXMLMap.insert( this.getXML_CompanyName(  ), sValue );
    sValue= new String( "Richmond, VA" );
    aXMLMap.insert( this.getXML_CompanyCity(  ), sValue );
aXMLMap.insert( this.getXML_Person_NODE(  ), null );
sName= new String( "Dick" );
    sLName= new String( "Douglas" );
    aXMLMap.insert( this.getXML_FirstName(  ), sName );
    aXMLMap.insert( this.getXML_LastName(  ), sLName );
```

Intelligent Advisor Code view

- The UML Factory Intelligent Adviser allows us to view the code generation results while designing the UML activity diagrams

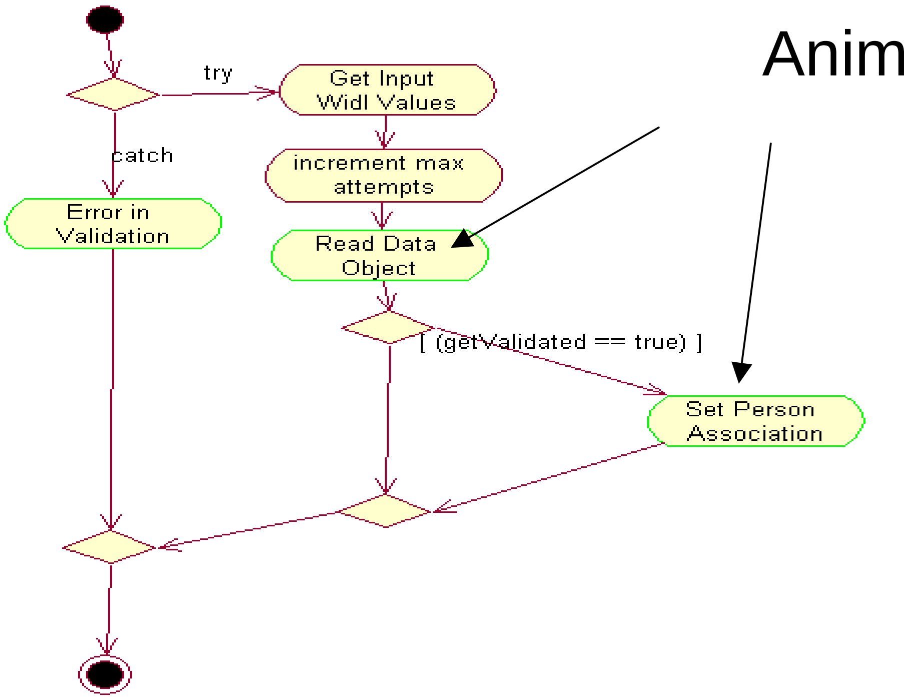


Rational Rose - PassageTrain.mdl - [Activity Diagram: SubmitValidate / SubmitValidate]
Diagram: LogonValidation.SubmitValidate

```
// declare references
boolean bValid=false;
String sUserName="";
String sPassWord="";
PersonCtl aPerson= null;
int ii=0;
System System= null;
// End declare references
try{

  //W Get Input Wdl Values
  // The input parameters returned in the Wdl come from the defined fields in the
  //

  sUserName = theWdl.getInputParameter( "UserName" ); // get user name from
  sPassWord = theWdl.getInputParameter( "Password" );
  this.setValidated( false );

  //W Increment max attempts
  ii = this.getCurrentTries( );
  ii=ii+1;
  this.setCurrentTries( ii );

  //W Read Data Object
  aPerson= new PersonCtl( );
  aPerson.setFirstName( sUserName );
  aPerson.setPassword( sPassWord );
  this.setValidated( aPerson.load( ) );
  if ( ( getValidated ( ) == true ) ) {

    //W Set Person Association
    this.setPersonCtl( aPerson );
  } // end if
}catch (Exception e0){
  e0.printStackTrace();

  //W Error in Validation
  System.err.println("LogonValidationPF.SubmitValidate Error in getting Person");
} // end try
```

Setting Animation Points

- Any activity within the activity diagrams can also have animation points defined. When the application is executing the activity diagram will display and select the activities with animation settings



Animation Points

Generating the Application

- After the static and dynamic elements of our application are defined we can organize these elements into UML Component Diagrams to define deployment definitions. From the components definitions we can then generate the complete Java class code, compile the code and package it into the appropriate application JAR

Collaboration Diagram XML Generation

- The collaboration diagram information for our use case definitions is also generated into an XML description document identifying the interface elements, XML mappings, and control process flow information. This XML document then becomes the basis for execution through the application
  - The XML document defines the process flow through the application
  - The engine that iterates the collaboration diagram XML document was also designed and generated from a UML model

Control Process Flow Generation

- Dynamic behavior represented within an application is designed within a control process flow class specification. The following sections will briefly illustrate the generation processing capabilities for dynamic behavior within a UML environment. The following code examples were completely generated from the UML model

Generating the UML Class Specification

- Looking at the State Machine implementation
  - Current State
  - Incoming Event
  - Synchronous Transitions
  - Asynchronous Transitions
  - Start and End States
- Packages, imports, declarations
  - Java package specifications can be explicitly or implicitly defined by the class locations within the UML model
- Association and Attribute implementations
  - Association implementations generate both the container and access methods for the stereotyped association. The type of container and resulting access methods are defined by the stereotype applied to the association between class specifications

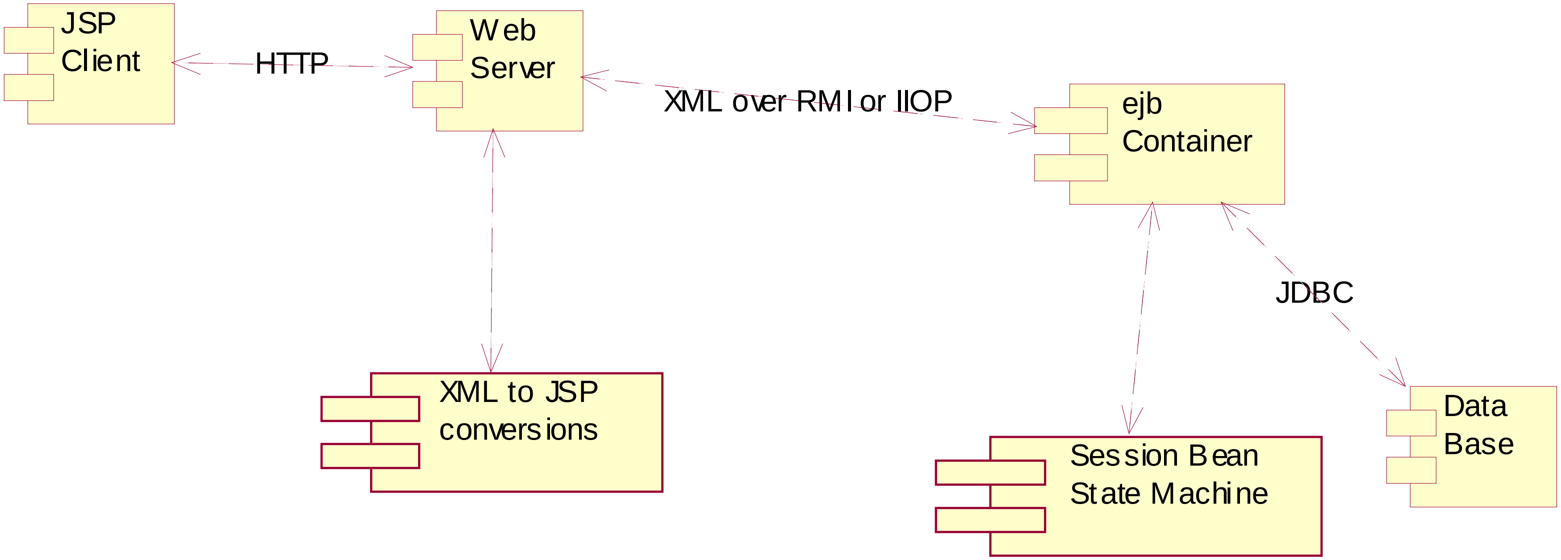
Method Implementation

- Activity Diagram Method Implementations
  - Activity diagrams are generated following the UML Object Notation. The activity generation can be viewed with either as a fully generated class or using the Intelligent Advisor code window
- Round Trip engineering
  - Roundtrip engineering allows developers to hand edit code generated from the UML Diagram. By Identifying elements which have been hand edited, the UML Factory generator will leave the method implementation as edited without forward generating from the model. The developer controls the granularly of roundtrip engineering
- UML Model Animation code
  - Animation points placed within the code can be turned off for individual points, the entire model being generated, and also ignored, from run time configuration properties

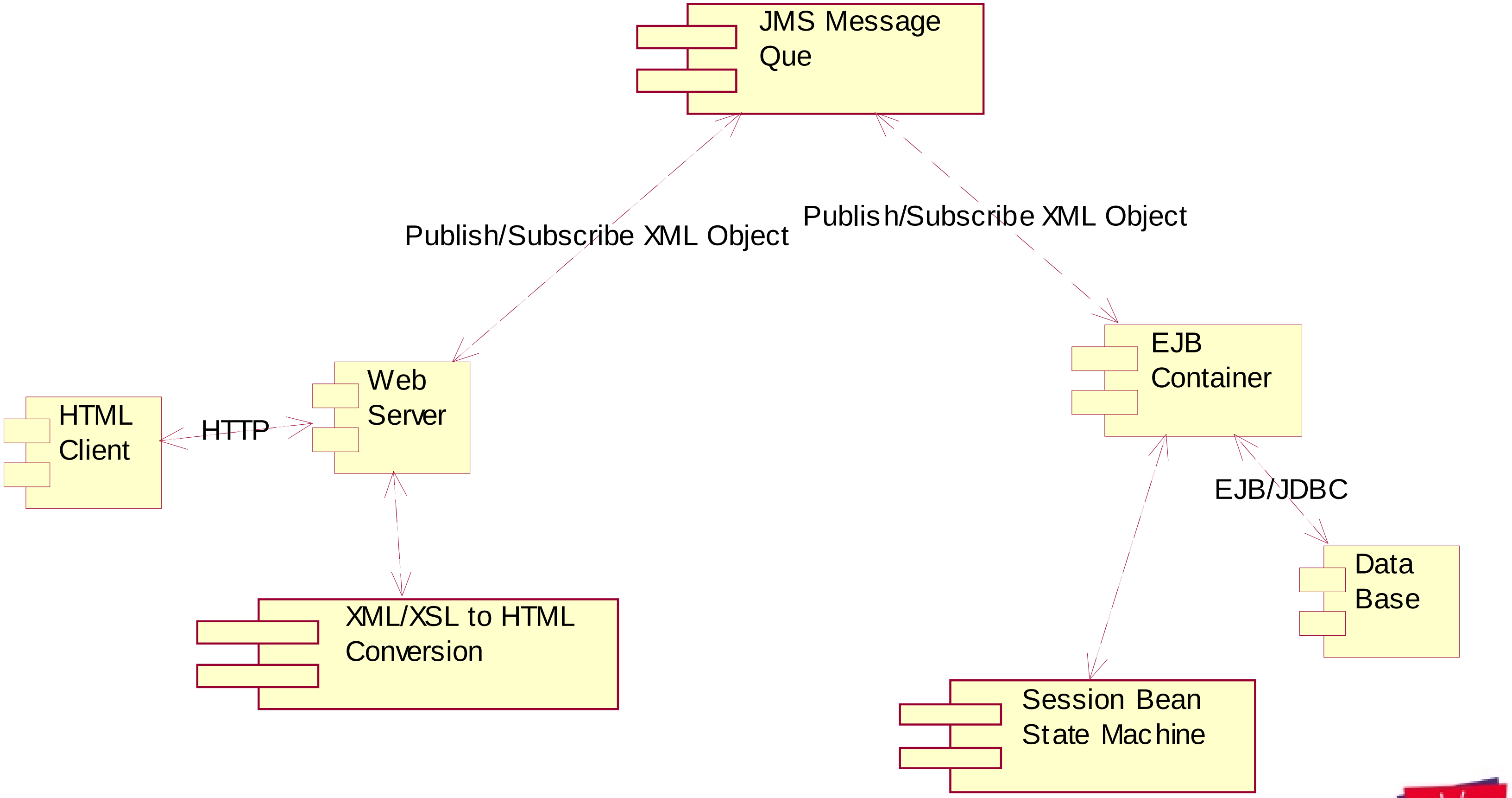
Executing the Architectures

- The following demonstrations will walk through executing the various architectures from the complete components on different types of application servers

Sample JSP™ Client to J2EE™ Application Container



Using the Java™ Message Service API (JMS) for Communications


JMS Message Que
Publish/Subscribe XML Object
Publish/Subscribe XML Object
Web Server
EJB Container
HTML Client
HTTP
EJB/JDBC
Data Base
XML/XSL to HTML Conversion
Session Bean State Machine

Summary

- This concludes the presentations and demonstrations for building n-tier enterprise applications with UML and XML data representations
- The demonstrations have illustrated the ability to completely model both static and dynamic application behavior within a UML model
- The UML model also defined components and deployment into multiple configurations
- The modeled demonstration illustrated the ability to use the same components within different J2EE™ platform-enabled implementations on multiple architectural environments

Enterprise Development Benefits

- Enterprise computing can improve the software development process by combining the descriptive power of the UML notation, and the flexible data representation and distribution capabilities of XML with the object oriented, network, and portable capabilities of Java technology
- Combining these technologies enables organizations to manage components increasing reuse, visibility, and implementation understanding; effectively reducing the complexity and total cost of ownership for deploying n-tier enterprise applications

JavaOne℠

Sun's 2000 Worldwide Java Developer Conference