



IBM Software Group

# **WebSphere® Process Server V6.0 WebSphere® Integration Developer V6.0**

## ***Business State Machines***



@business on demand.

© 2005 IBM Corporation  
Updated December 19, 2005

This presentation will provide an overview of WebSphere Process Server V6.0 and WebSphere Integration Developer 6.0 Business State Machines.

## Goals

- Introduce Concepts of Business State Machine
- Introduce the Architecture and elements of the Business State Machine
- Present the client programming model used for interacting with Business State Machines
- Describe Development, Packaging and Deployment of Business State machines



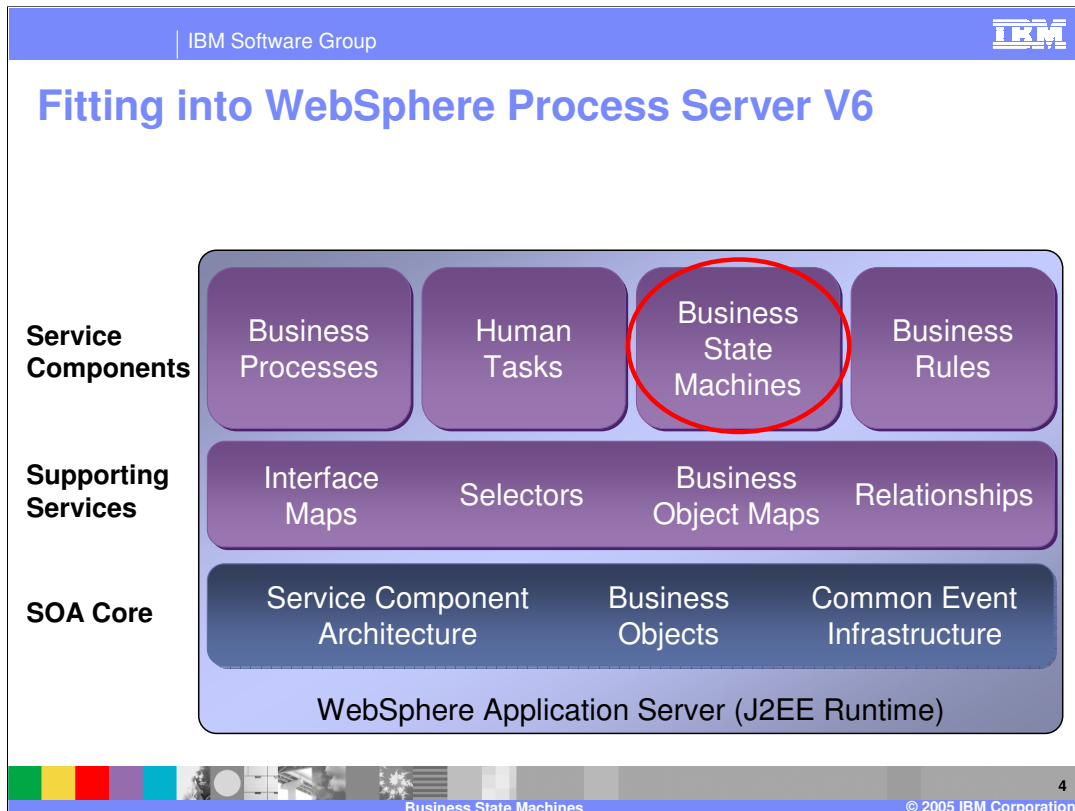
The goal of this presentation is to introduce the concept, architecture, and elements of a business state machine and discuss development, packaging, and deployment. The client programming model will also be covered.

## Agenda

- **Overview**
- Architecture / Big-Picture
- Details
- Putting it Together
- Example
- Problem Determination
- Summary and References



This section will provide an overview of business state machines.



This graphic represents the WebSphere Process Server Architecture stack.

The focus here is on the Business State Machine as a Service Component. The Business State machine is an SCA component, which by design takes advantage of the underlying infrastructure, such as maps, relationships, Business Objects and the Common Event Infrastructure.

The Business State Machine is a kind of implementation for implementing business logic with the Service Component Architecture. There is a special editor used to build the State Machine and when it is finished, it can be added to the module assembly where it can be hooked up with other SCA components.

## State and State Machines

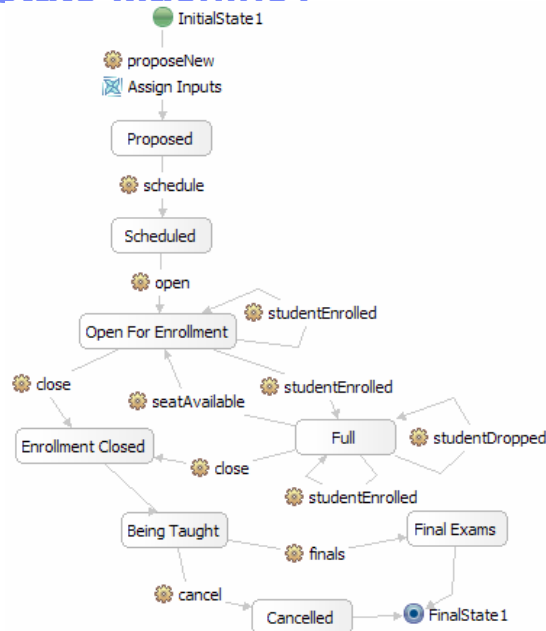
- A 'State Machine' model is a way to describe the **dynamic behavior** of an application or business process by focusing on the events that cause a transition from one state to another.
- A 'Business State Machine' is an implementation of a business model that 'executes', moves from state to state, based on events received.



A business process behaves differently to given inputs over time based on its current state, which is determined by the values of one or more of its properties, such as data. This behavior over time provides a clear and concise understanding of how the business process will respond to a given input event and is referred to as dynamic behavior.

## What is a Business State Machine?

- An efficient way to model event-based processes
- States and state transitions frame the process
- Logic embedded in the transitions
- Based on UML 2.0 State Machine



This diagram is a model that describes the business process for proposing, scheduling and completing a new course. Each rounded rectangle represents a different state in the life cycle of the business process. Each arrow between the states represents a possible transition from one state to another and each transition is triggered by an event, represented by the 'gear' icon.

The business process is viewed in terms of the possible states and the events that cause the business process to move from one state to another.

Initially this business process is very sequential. The initial proposal or event that moves the process to the 'Proposed' state is received, at which time the faculty and the administration can determine if this new course will be appropriate for their needs. The decision is then made to schedule the class so the 'scheduled' event is sent to the business state machine and it will then move to the next state, which is 'Scheduled'.

According to this business process, this course will remain in the 'Scheduled' state until someone sends the 'open' event to the business state machine to notify it that it is time to start accepting enrollments. At this point, there are several possible events to consider and the transition that will be invoked depends on the current state of this business state machine instance. For example, the number of students currently enrolled.

Laying out the business process this way, it becomes apparent that there are missing conditions and transitions. For instance, how does the state machine move from the proposed state if the proposal is rejected? How can the students be notified that the class is now open for enrollment? How can the class be canceled if there are not enough students? What are the rules that determine the criteria for canceling a class?

The power of modeling the business process as a state machine is that these questions are brought to the surface and a place is provided where you can specify the newly discovered information.

## When to use a Business State Machine?

- Use Process Choreographer when:
  - ▶ steps in a process tend to happen in sequence
  - ▶ some event handling and looping is ok
  
- Use a Business State Machine when:
  - ▶ the business process is heavily event-driven
  - ▶ the reaction to these events is dependent on the process state
  - ▶ the process may revert to prior states
  - ▶ some sequential steps ok



A question that always arises when talking about WebSphere Business State Machines is considering WebSphere Process Server has two different ways to specify a business process, why not just use the Process Choreographer to specify a BPEL business process?

First, you might already have your business process defined in terms of a state machine model. If this is the case, this will be a very natural and easy approach for you.

If this is not the case, then take a look at the business process, and as you describe it, does it flow naturally in a sequential path with a few branches to sub processes, waiting for input at a pick, making a decision at a switch, invoking some service, waiting for some human interaction and then ending with a reply? If this is the case, then the Process Choreographer will be the natural choice for developing your business process and collaborations.

On the other hand, if you notice that you need to start creating loops, interjecting complex logic to go back to a prior state or maybe the entire process is driven by events coming into the system, for example with lots of picks everywhere, then you should consider the state machine model.

When considering the Business State Machine, you want your incoming events to be one-way messages and you do not want to have human interactions directly on the transition. Although you can initiate work as part of the transition, you want to get to the next state as soon as possible and wait for input there.

Combining BPEL business processes created using Process Choreographer with Business State machines is discussed later in this presentation.


## Agenda

- Overview
- **Architecture / Big-Picture**
- Details
- Putting it Together
- Example
- Problem Determination
- Summary and References











This section will cover the Architecture of the Business State Machine.



IBM Software Group 

## Elements of a State Machine

- Transition
  - ▶ Events (Operations) 
  - ▶ Guard (Condition) 
  - ▶ Action 
- State
  - ▶ Entry 
  - ▶ Exit 
- Composite States 

  9

Business State Machines © 2005 IBM Corporation

The elements of the state machine are shown here. Each element will be discussed in detail but for now just take note of the icon and the relationships between the elements.

A transition connects one state to another in a state machine. It is used to control the transition from one state to the next by recognizing the triggering Event (operation), evaluating the Guards (or conditions) necessary for execution to flow through it, and determining what Actions can occur should execution be allowed.

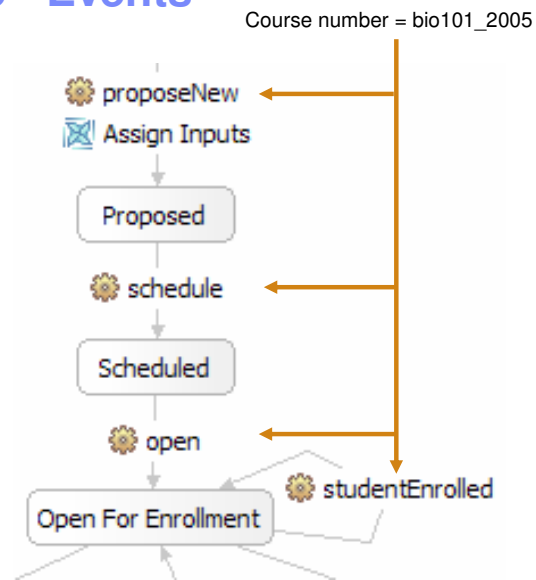
Events, Guards and Actions are all sub-elements of the transition. They are optional features that can be used on a transition.

Entry and Exit are sub-elements of the state. They are optional opportunities for initiating (or invoking) work that you may need to do upon entry to or exit from a given state.

The Composite State is a way of simplifying complex state machines by using the composition pattern.

## Business State Machine - Events

- **Events** can cause a transition out of a state
- **Events** map to the **operations** that are available on the state machine SCA component
- Business Objects or primitives can be used as **input/output parameters**
- Part of the input parameter is used to **correlate** to a state machine instance



This graphic depicts what an event is and what it does.

The Business State Machine is just another kind of SCA component and as such the interface is defined using WSDL and the data are defined using XML Schema. This means that the events sent to the business state machine will be defined as operations on a WSDL port type definition, for example interface. The events (operations) should be short and concise, with one-way operations preferred.

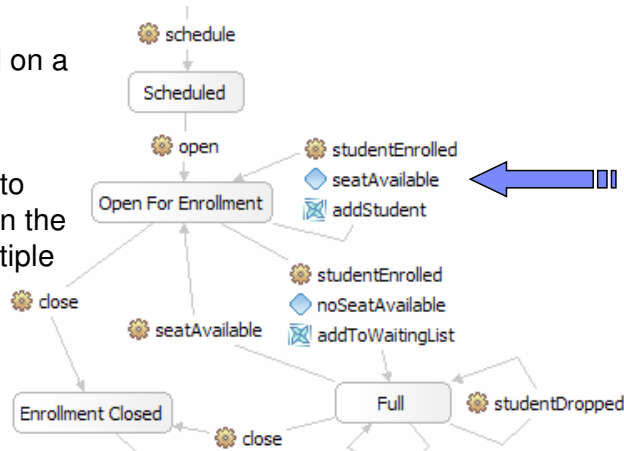
The business process will wait at a given state until one of the events it is waiting for arrives. The events it will be waiting for are the ones on the outgoing transitions.

The events (operations) must have at least one parameter on the input. This would contain the correlation data, which is equivalent to a handle to a given business process instance.

The diagram is a model, like a template, for many possible instances of this business process. Each instance will have a unique identifier ( a correlation id) and unique values for its properties. In this case the correlation id might be something like the course number with the date. For example, bio101\_2005. The correlation id would be supplied as one of the parameters to the *proposeNew* event, which would then cause the instance of the business process to be created with this unique identifier. The unique identifier would then be used by the other events to reference this particular instance. This means that all other events must have the course number as an input parameter.

## Business State Machine – Guards (Conditions)

- A **Guard** is implemented as a condition on the transition. A guard can restrict what transitions will occur based on a Boolean condition.
- A Guard (**condition**) can determine which transition to take, especially useful when the same event can cause multiple transitions
- Logic can be defined via
  - ▶ Visual Editor
  - ▶ Java™
  - ▶ Service Invocation



You might have noticed in the earlier discussion that the “OpenForEnrollment” state has at least two outbound transitions. The question of whether they both get fired or which one gets fired first arises.

Only one transition can be traversed and the state machine can only be in one state at a time and can only go to one state. Therefore, based on the current state of the business process, there must be a way to determine which transition to take. This is done with the Guard, sometimes called the condition or Guard condition.

In the example of ‘OpenForEnrollment’, there are a couple of variables internal to the business state machine called `MaxStudents` and `CurrentlyEnrolled`, which will be used by the logic of the Guards to determine if the class is full or not. The variable for `CurrentlyEnrolled` will be updated whenever a student is successfully enrolled and in a complete solution this would also be updated when a student dropped their enrollment from the class. The `studentDropped` event has not been modeled yet.

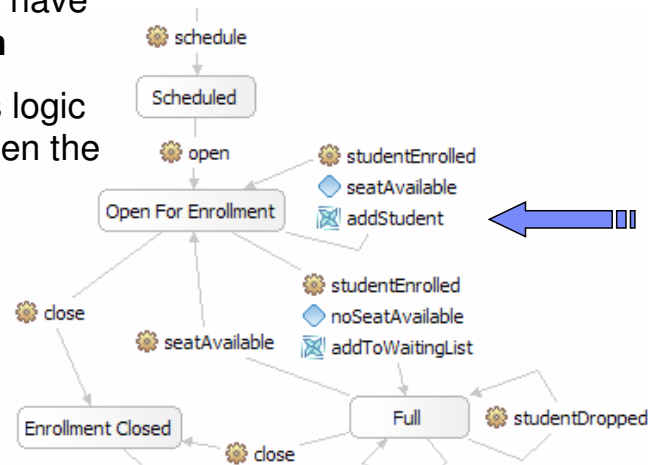
The Guard is the place where you specify the logic used to determine if this transition is the one you want at this time. The logic can be specified using the Visual Editor tool, which has visibility to the global data and input parameters for the state machine. The visual snippet is converted to inline Java™ code so if you are already adept with the SCA client programming model and are comfortable with writing the Java™ code directly, you can. Or you can invoke another service. These 3 mechanisms for specifying logic or actions are used throughout the WebSphere Integration Developer authoring tool.

Invoking a service is by far the most powerful means, because you can invoke any SCA component, including a Business Rule. However, it will be more costly in terms of performance when compared to inline Java™ code.

Another advantage to using the Service Invocation is that it can easily be emulated using the WebSphere Test Environment, the Component Tester.

## Business State Machine - Actions

- State Transitions can have an associated **Action**
- An **action** represents logic that can be called when the transition occurs
  - ▶ Visual Editor
  - ▶ Java
  - ▶ Service Invocation



Actions allow you to initiate some work as part of the transition.

It could be that as you move from one state to the next that you need to

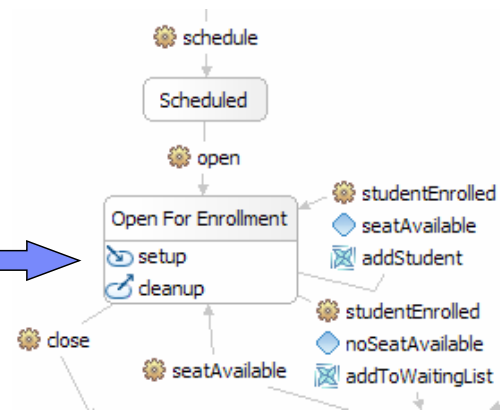
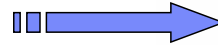
1. notify someone
2. initiate another sub-process, such as a BPEL process
3. provide a return value for the transition event, if it is a 2-way event (operation)

The tools used to create and invoke your application code are identical to what was available with the Guard, namely high level visual programming, Java and invocation of existing services.

You do not want to linger in the transition too long. This means that if you call out to another service, you want it to return as soon as possible. In this example a student is added to the waiting list. This could be very simple, involving the update of a few global variables associated with the state machine, or it could be more involved, requiring an update to an external database and even involving some human interaction. If human interaction is involved, creating a long delay, the model should be changed to introduce one or more new states and events.

## Business State Machine - Entry/Exit Actions

- **Entry** logic can be called upon entering a new state
- **Exit** logic can be called upon exiting a state
- Independent of which transition enters or exits the state
- Logic can be defined via
  - Visual Editor
  - Java
  - Service Invocation



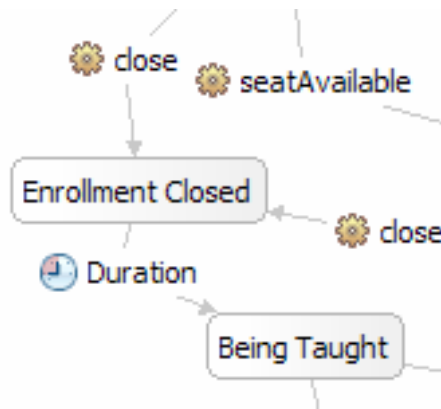
Entry and Exit Actions are places to perform work that might need to be done every time a given state is entered or exited.

It could be that upon entering a state, you need to verify the availability of some external resource and if it is not available, initialize it.

When leaving, you will need to check to see if you are the last one using the resource and free it up if that is the case.

## Business State Machine - Timers

- A transition can be caused by a timeout
  - ▶ Durations (1 day)
  - ▶ Expirations (Dec 31, 2005)
- Don't wait forever for someone to cancel



Timers prevent the possibility of getting stuck in a given state.

Timers can be used in conjunction with Guards and Actions or on their own as shown here.

They cannot be used with an Event because they are essentially doing the same thing. For example, initiating the transition.

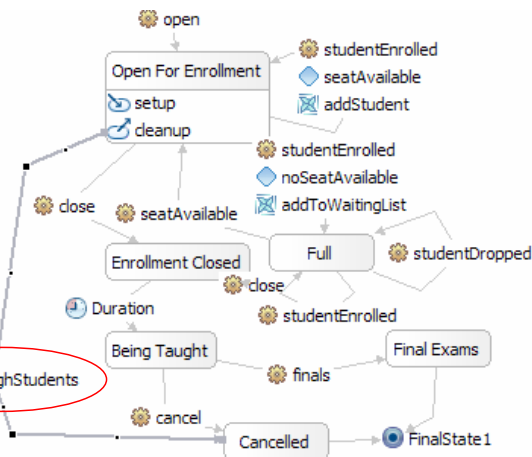
Durations use the default WebSphere calendar, which allows you to specify a period of time in Years, Months, Days, Hours, Minutes and Seconds, after which the timer will be fired, causing the transition event.

Expirations allow you to set a specific date when the timer will be fired, causing the transition event.

Timers and durations can be specified using the Visual Editor, inline Java programming or literally using the input panels available in the details tab.

## Business State Machine - Automatic Transitions

- If a transition has no event (operation) or timer, it is considered an **automatic** transition
- **Automatic** transitions enable sequential processing
- **Automatic** transitions in conjunction with conditions can provide default behavior.



Automatic transitions do not have a triggering event associated with them. This means that the state machine engine automatically fires the transition if the Guard condition is met.

If you have an automatic transition without a Guard condition there cannot be any other outgoing transitions. The Business State Machine Editor will flag this condition.

If you have an automatic transition with a Guard and a Timer and the condition is false when the Timer fires, the transition will not occur.

## Agenda

- Overview
- Architecture / Big-Picture
- **Details**
- Putting it Together
- Example
- Problem Determination
- Summary and References

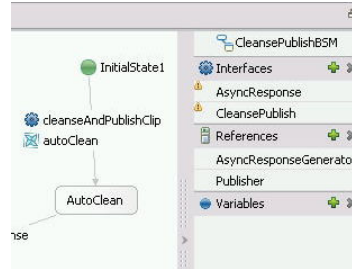


This section will provide details concerning the Business State Machine.



## Interfaces, References and Variables

- A Business State Machine is a kind-of SCA component.
  - ▶ Interfaces define the Events (operations)
  - ▶ Actions can reference (use) other SCA components.
  - ▶ Business Objects can be defined and assigned to instance variables
  - ▶ Instance variables are maintained with the state machine.



Order	
ID	string
Description	OrderDescription
orderDate	date
discountRate	decimal
total	decimal
company	Company
customer	Customer
shippingMethod	string
state	string
Orderline	Orderline

Because the Business State Machine is another SCA component, it uses the same tools and editors for defining the Interfaces and Business Objects.

Before you start to build your State Machine, the interface, including the types of business objects that will be passed around, should be defined.

You could have multiple interfaces, but be sure that you can extract the correlation information from any event that will be used.

The screen shot on top shows the interface in the Business State Machine Editor for associating the Interfaces for incoming events (operations) and the References for outgoing interactions with business partners.

This interface is also where you can create and add variables, which are SCA business objects.

If you use this interface to create a new business object, you will need to edit the business object at a later time to add the attributes.

IBM Software Group IBM

## Business State Machine – Referenced Partners

- Partners represent external services (SCA components) that are called by the state machine
- Can be called from
  - ▶ Actions
  - ▶ Entries
  - ▶ Exits
  - ▶ Guard

18  
© 2005 IBM Corporation

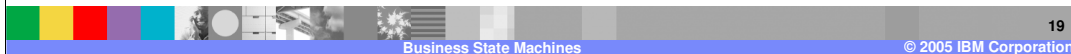
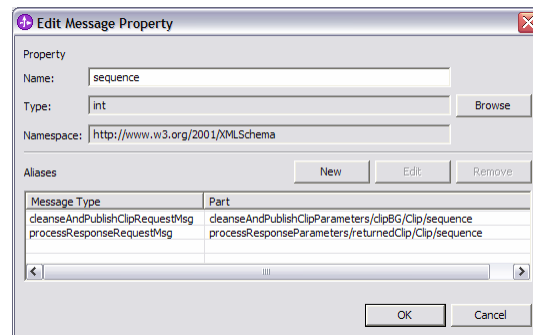
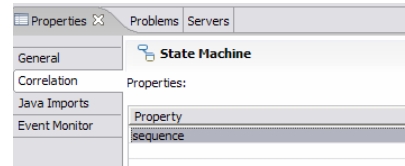
When the Business State Machine needs to collaborate with another service you must first create the reference that will make the association between the business state machine and the other entity. In this case the reference is given the name, 'Publisher'.

Then when the reference is used, whether from an Action, Entry, Exit or a Guard, select the invoke radio button, pick the reference and operation and setup the assignments to the parameters for the input message and, if available, the output message.

Because all of this is based on selecting existing entities, you can see why it is important to have the interfaces, variables and business objects defined ahead of time.

## Correlations / Properties / Aliases

- A correlation is used to identify unique instances of the Business State Machine.
- A correlation is a property with one or more message aliases which are used to get and set the value of the property at runtime.
- The aliases map the messages for the Events (operations) to the property.



A correlation is the handle or ticket to an existing instance of a specific Business State Machine.

This is managed by creating a global property (data element), the top part of the **'Edit Message Property'** dialog.

The type of the property must be a simple type, for example, string, int, or date.

Next, one or more aliases are created to associate the messages that will be used by the business process engine, to set the value of this property at runtime. There must be an alias for each incoming event to the business state machine, that is all the events on the interfaces.

Creating the alias is done in the bottom part of the dialog.

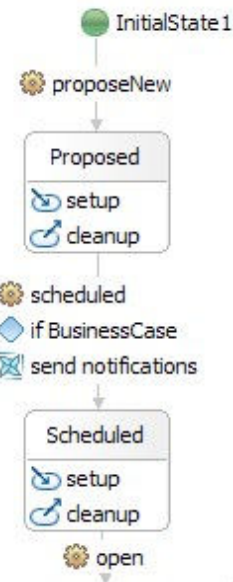
The BPEL business process engine that is actually executing the business state machine at runtime.

This is a simplification of the correlation feature which is a part of the Process Choreographer editor. Here only a single correlation can be created that will be used for all incoming events. There is no need to create a correlation for outbound collaborations when working with a business state machine.

Composite correlations, which are correlations with several properties, can also be created.

**Hint:** when defining the business objects to be used in the events for your business state machine, specify 'required', for the fields to be used in the correlation.

## Overall Business State Machine Flow



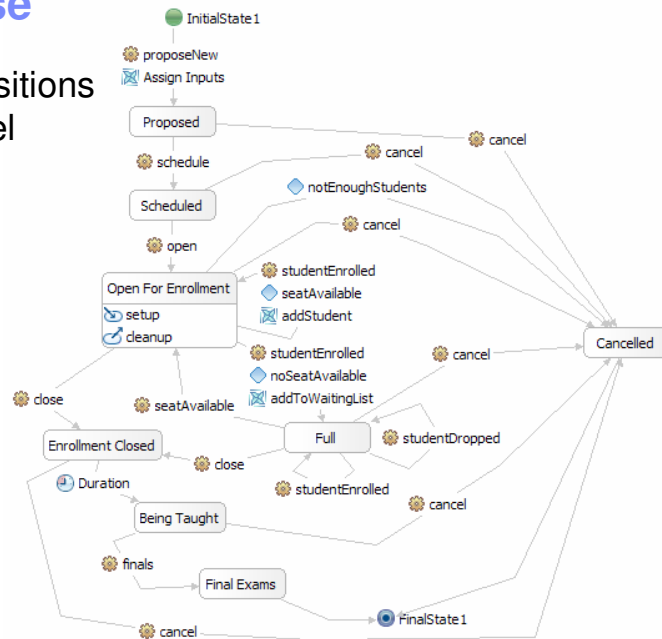
- Use **input parameter**, such as the customer number or purchase order number for **correlation** to get the proper **instance** and **current state**
- For each outgoing transition that supports the event, check the **Guard** conditions (if specified) for a **true** condition
- Process the state **Exit Action** (if specified)
- Process the **transition's Action** (if specified)
- Change the **State**
- Process the new state's **Entry Action** (if specified)
- Check for any **automatic transitions** out of the new state and repeat, or wait for next event

The overall flow, starting from the 'Proposed' state is as follows:

1. The 'scheduled' event is received by the 'business process engine'
  - The correlation information is extracted from the input parameters using the correlation aliases and used to lookup the business process instance in the BPEDB.
2. For each outgoing transition, check the Guard conditions for one that returns true. (if you have more than one outgoing transition, they must have conditions. )
3. If the condition is true then process the Exit action for the current state.
4. Process the Action on the transition, if one is specified.
5. Change to the new state
6. Process the Entry Action for the new state, if one is specified
7. Check for and process any automatic transitions.
  - This might entail checking any Guard conditions that are defined.
8. Wait for the next incoming event.

## The Cancel Case

- Notice all the transitions going to the cancel state.



21

Business State Machines

© 2005 IBM Corporation

An important consideration is what to do when the diagram becomes unwieldy.

In this case, allowing for the cancellation of the class at anytime in the life cycle of the business process results in a very messy diagram with a transition from every state to the cancel state. This starts to detract from the description of business process and shifts the focus to laying out the diagram.

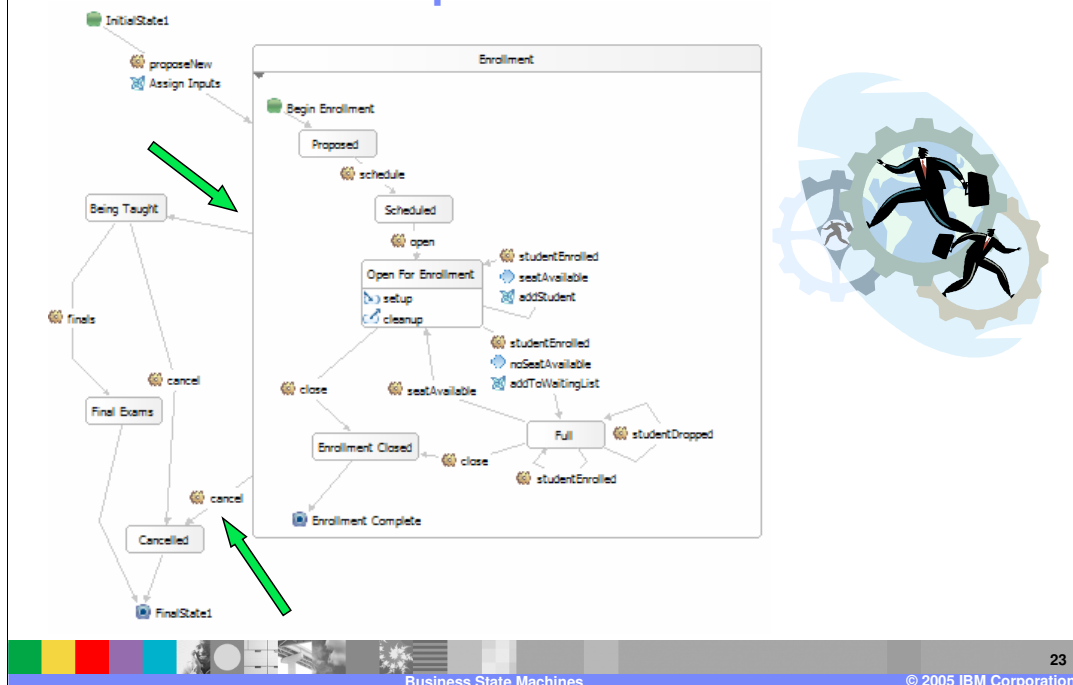
## Composite Business State Machines

- A Business State Machine can be simplified by using the Composite pattern.
  - ▶ A business state machine nested in a business state machine.



The solution is to use the Composite pattern, a state machine within a state machine.

## Cancel with a Composite



Here, the heart of the business process is defined inside a composite state. The composite state is an element in the state diagram you are currently working on. You must edit it directly. You do not create a separate state machine and drop it onto your current diagram.

Focus your attention on the transitions leaving the composite state. (the 2 green arrows)

1. The Automatic transition on the top is the transition taken when the composite reaches it's internal 'final' state, labeled 'Enrollment Complete'. When the composite has a final state, the automatic transition is required.
2. Notice the cancel transition stemming from the composite on the bottom. The semantics of this is such that all of the states inside the composite can receive the cancel event.
3. Finally, notice that you can also have Entry and Exit Actions associated with the Composite state as well. They would be the place to initialize the class roster and the waiting list for this enrollment process.

## Agenda

- Overview
- Architecture / Big-Picture
- Details
- **Putting it Together**
- Problem Determination
- Summary and References



This section will pull together all the topics discussed thus far.



## Constructing a Business State Machine

- Model your Business State Machine
  - ▶ e.g Define the states and data required
  - ▶ Identify the business related attributes to be used as the correlation identifier.
- Define the Business Objects that will be used by your state machine.
- Define the Interfaces that will be the *Events* for the transitions.
  - ▶ Implement as one-way operations when possible.
  - ▶ Two-way operations are required if you need to return fault information and they should return promptly.
  - ▶ Actions on *Transitions* should also be one way operations when possible. Two-way operations should return promptly.
- Actions on *Entry* and *Exit* should be used as constructor and destructors and avoid calling out to other processes that can introduce side effects.
- Identify any other SCA components or business partners you will need to collaborate with before you start to layout the business state machine.

When designing and developing Actions for a Business State Machine, the work can be done in another process. If there is some result that will affect the state of the overall business process, this state should be modeled as a discrete state in the BSM.

Do not get caught in waiting for a return from an action.

When using Two-way operations for the Actions, the scope is for the transition.

You can use an Action on the transition to assign values to the parts of your output message.

Two-way operations are necessary if you need to verify that the operation was received and completed successfully or if you need to catch faults.

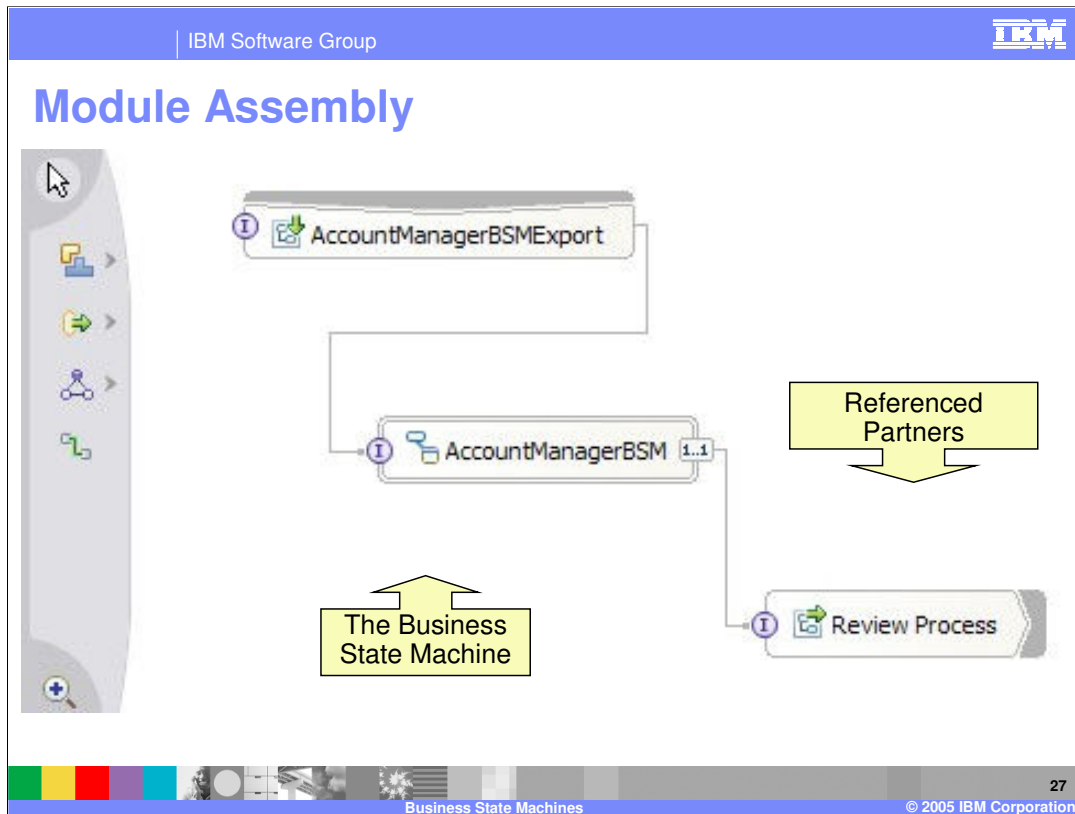
## Constructing a Business State Machine

- With all the components in hand
- Create a Module
- Create the State Machine using the interfaces and objects
  - ▶ e.g. lay it out using the State Machine editor.
- Add the State Machine to the Module Assembly
- Wire up the references and define the Exports.

**note:** For step by step how-to instructions,  
see the lab exercise.



The steps to create a business state machine are shown here.



This is standard SCA construction.

If you make changes to the Interface or References to the Business State Machine, you must delete and recreate the relevant components in the Assembly diagram.

## Running the Business State Machine

- Make sure the BPEL container is installed and configured.
  - ▶ Since the implementation created is a BPEL process all the development considerations for BPEL processes will apply.
    - The development flag in the server configuration
    - Removing the application from the server
- A couple of ways to interact with your State Machine within the WebSphere Integration Developer.
  - ▶ WebSphere Test Client (WTC)
    - First Class test environment in WebSphere Integration Developer (WID)
  - ▶ APIs for calling an SCA component from Java code.
- Alternately, export the EAR, install/deploy it to the WPS 6 runtime, where you can test and run it.



This slide shows what gets generated when you save your diagram.

The state machine that you create is implemented as a long-running BPEL business process. You might want to look at the generated BPEL. Once you do, you will appreciate all the work that has been done for you behind the scenes.

When it comes to deploying, testing and administering your business state machine, its good to know that it is really a BPEL business process. If you are not familiar with the administration issues for BPEL business processes, now is the time to become familiar.

- In a production environment, where the development flag is false, you must stop or complete and terminate all running instances before you can remove the J2EE EAR.
- To manage the situation where you need to deploy a new version and allow the existing ones to complete, there is the 'valid From' time stamp, which you can configure in the details section for the Business State Machine. This timestamp will set a date and time at which all new business state machines of this type, will start to use the newer version, providing an opportunity for the existing ones to complete normally.
  - Once all the existing business state machine instances of this type have completed and been deleted, the older version can be removed.

The WebSphere Test Client is sometimes referred to as the Module or Component Tester. This is the way to conduct the unit test for a business state machine. It is very easy to drive a business state machine through its states using this tool.

## Agenda

- Overview
- Architecture / Big-Picture
- Details
- Putting it Together
- **Problem Determination**
- Summary and References



This section will cover problem determination.

## Problem Determination

- Trace strings
  - ▶ Use the traces to get valuable and detailed information regarding the component your working with
  - ▶ com.ibm.bpe.\*
- Log messages
- Event Monitoring
  - ▶ CEI events can be used to track the state of the business state machine
- The Process Debugger



Available resources and options for problem determination are shown here.

## Best Practices

- Best practices requires a thorough understanding of how the Business State Machine works in the context of SCA and BPEL.
  - ▶ It is too early in the product life cycle to provide this kind of information.
- Think in terms of events
  - ▶ Fire and forget
- Avoid long running synchronous collaborations.
  - ▶ If you need to 'call out' to another SCA component or Web Service make it asynchronous so the state machine can be ready for the next event.
- Make sure the development flag is set in the server you're using for development.



Best practices for Business State Machines are outlined here.

## Agenda

- Overview
- Architecture / Big-Picture
- Details
- Putting it Together
- Problem Determination
- **Summary and References**



This section will provide a summary and references.



## Summary

- A Business State Machine is way to model complex dynamic behavior of business systems.
  - ▶ Event driven
  - ▶ Involves returning to previous states
- A Business State Machine can also be used to tie together other business processes in new scenarios.
  - ▶ As a controller or router.
    - Actions can be used to invoke other business processes.

## Trademarks, Copyrights, and Disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM	CICS	IMS	MQSeries	Tivoli
IBM (logo)	Cloudscape	Informix	OS/390	WebSphere
e(logo)/business	DB2	iSeries	OS/400	xSeries
AIX	DB2 Universal Database	Lotus	pSeries	zSeries

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

Microsoft, Windows, Windows NT, and the Windows logo are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

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

Linux is a registered trademark of Linus Torvalds.

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

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements and/or changes in the product(s) and/or program(s) described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (e.g., IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing  
IBM Corporation  
North Castle Drive  
Armonk, NY 10504-1785  
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

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

Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.