IBM Software Group

WebSphere® Commerce V6.0

Payments technical overview

business on demand.

© 2008 IBM Corporation
Updated June 20, 2008

This presentation covers the WebSphere Commerce version 6 payments technical overview presentation.

This presentation will provide an in-depth technical view of the payments component for WebSphere Commerce version 6

Agenda

- Role of payment component in shopping flow
- Overview of the V6 payments architecture
- Overview of the payments processing flow
- Step by step example of payment processing

© 2008 IBM Corporation

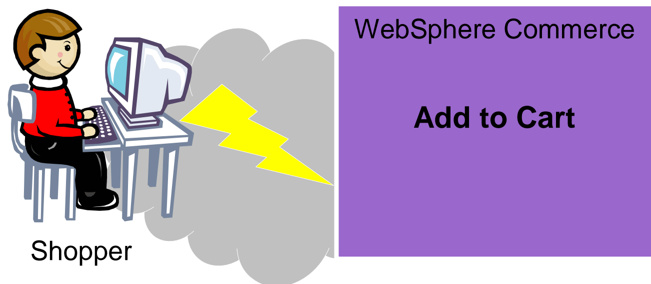
This presentation will begin by giving an overview of how the payment component plays a role in the WebSphere Commerce shopping flow.

Next, it will outline the WebSphere Commerce version 6 payments component architecture and discuss the Payments processing flow.

To illustrate this, you will see a step-by-step example of the payment processing flow.

Payment shopping flow

1. Shopper builds a shopping cart
   - Add, remove, update



© 2008 IBM Corporation

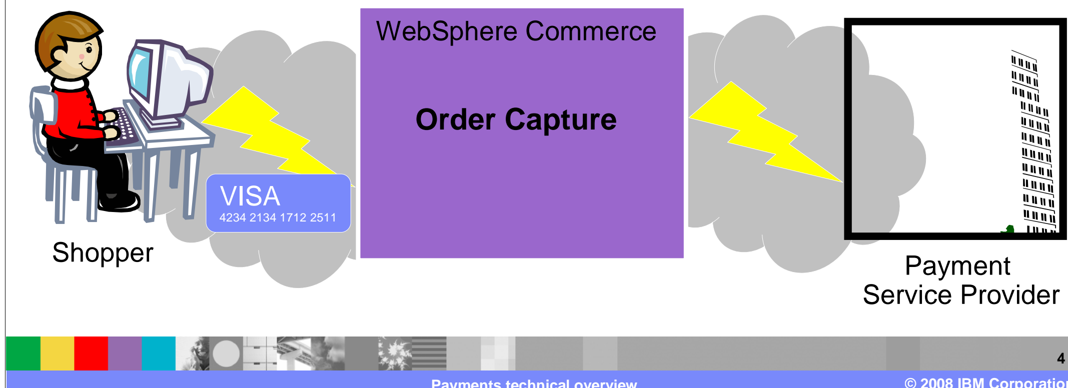
The next slides will demonstrate how WebSphere Commerce can handle payment transactions in the context of the order flow.

The first step in the order flow is for a shopper to build a shopping cart. This is handled by the order subsystem commands such as add, remove, and update.

Payment shopping flow

2. Shopper submits their order

- Order capture process is invoked on the server
- Shopper provides payment data
- Server approves payment with Payment Service Provider



© 2008 IBM Corporation

The shopper submits their order.

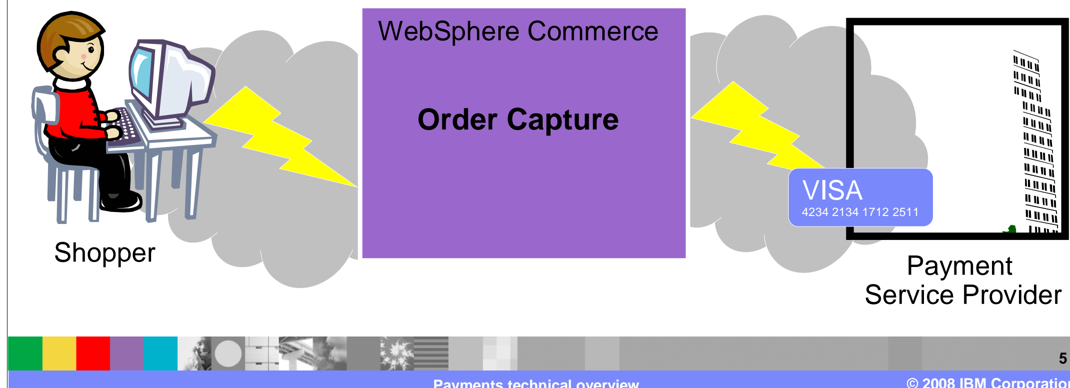
The order capture process is invoked on the WebSphere Commerce server.

When submitting an order, the shopper provides payment details for their shopping cart. This information makes up the payment record.

Payment shopping flow

2. Shopper submits their order

- Order capture process is invoked on the server
- Shopper provides payment data
- Server approves payment with Payment Service Provider



During order capture, WebSphere Commerce can be configured to validate and approve the payment against a backend payment service provider.

The payment service provider is a broker between a store or merchant, and the financial institutions, such as credit card companies and banks.

IBM Software Group

Payment shopping flow

3. Order is released to fulfillment
   - Payment authorization or deposit can occur at this stage in the order life cycle depending on your payment business policy

Once the payment is approved and the inventory is allocated, the order life cycle releases the order to fulfillment.

This step might require a transaction with the payment service provider. For example, the payment might need to be authorized before the item can be shipped, or depending on the payment method, funds might be deposited.

Payment shopping flow

4. Order is shipped
   - Server finalizes the order
   - Server deposits the funds for the order

© 2008 IBM Corporation

Once the order has been fulfilled by the fulfillment center, the order is shipped.

Within WebSphere Commerce, the order is finalized.

This finalization typically includes depositing the payment, which requires interaction between WebSphere Commerce and the payment service provider.

Payment shopping flow

1. Shopper builds a shopping cart
2. Shopper submits an order
   - Payment information is captured and stored
3. Order is released to fulfillment center
   - Payment is typically authorized/approved before releasing the item to be fulfilled
4. Order is shipped
   - Payment is deposited/captured

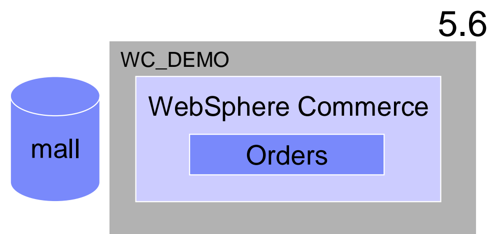
- The Payment transactions above are communications between the store and a Payment Service Provider

© 2008 IBM Corporation

In summary, there are specific points in the Order life cycle in which there is an interaction with the backend payment service provider.

This slide summarizes the interaction between the order and payment subsystems in WebSphere Commerce.

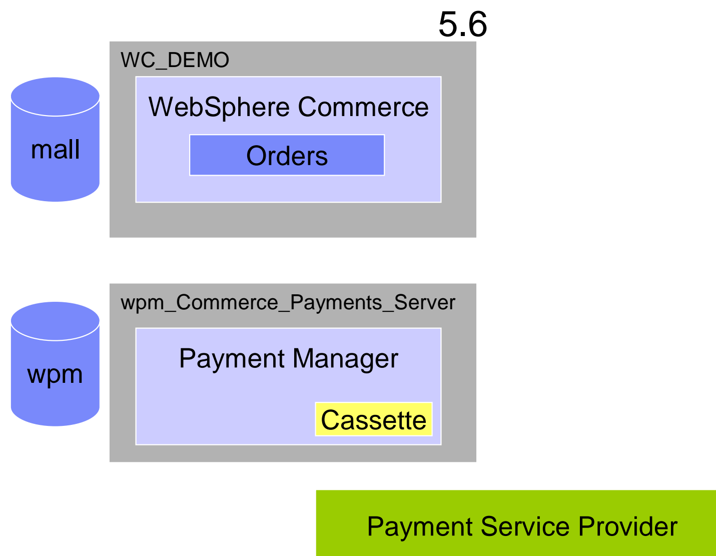
Architecture



© 2008 IBM Corporation

The next few slides will compare the architecture of WebSphere Commerce Payments from previous releases so you can see what the differences are in version 6.

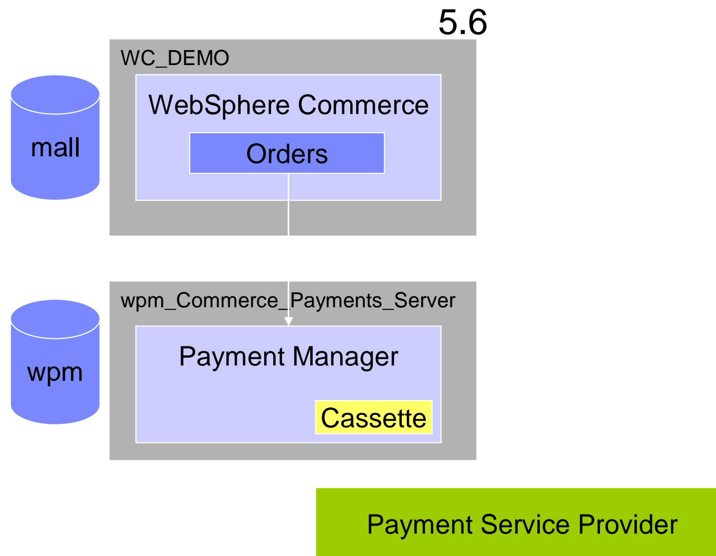
In version 5.6, there is an orders component that runs within the WebSphere Commerce Application Server and uses its own database.



To process payments, a WebSphere Commerce payment manager server needs to be created.

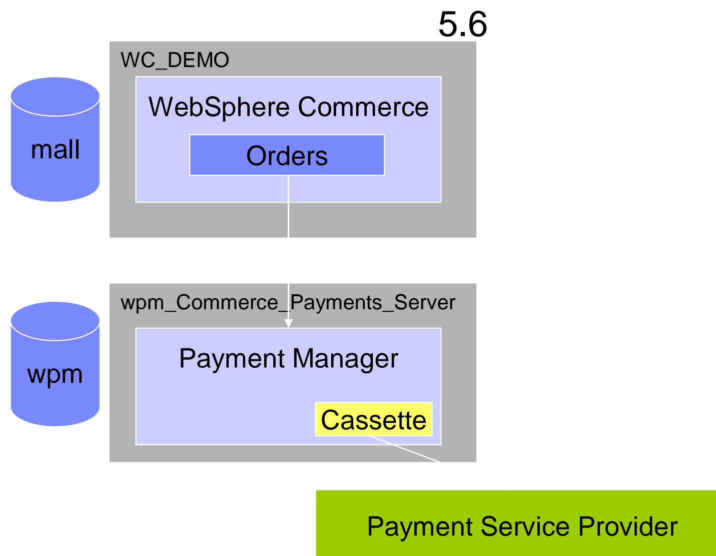
The payment manager server runs in a separate application server using an independent database.

The payment server is then used to interact with the back-end Payment Service Provider.



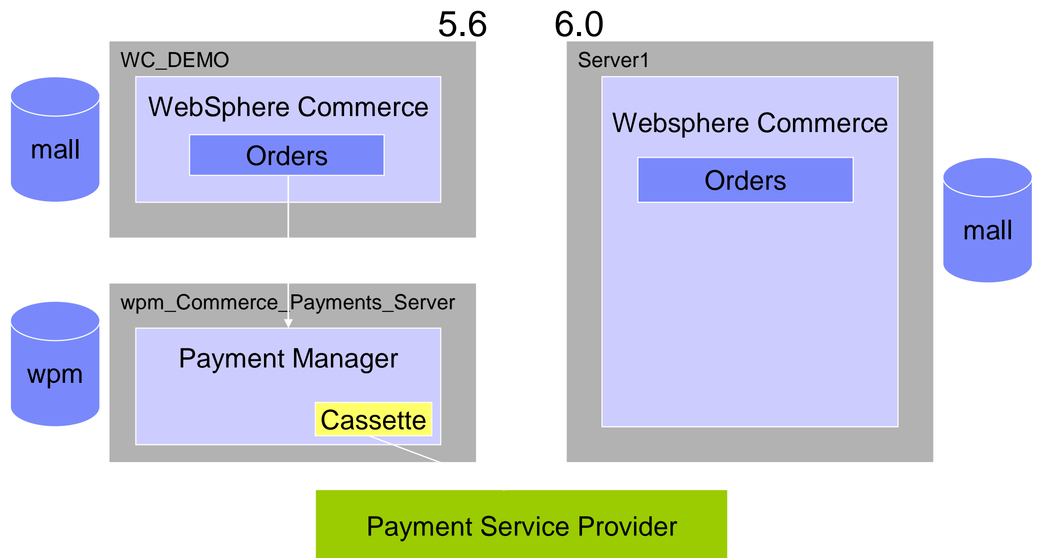
The request flow begins with the WebSphere Commerce Orders component making a remote call to the Payment Manager.

Architecture



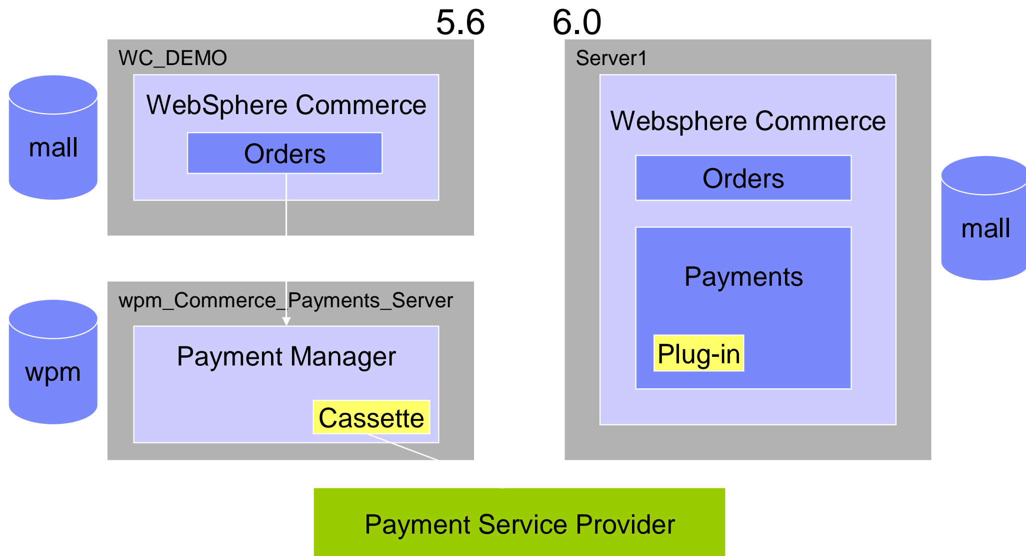
© 2008 IBM Corporation

The payment manager uses a cassette to initiate the transaction with the payment service provider.

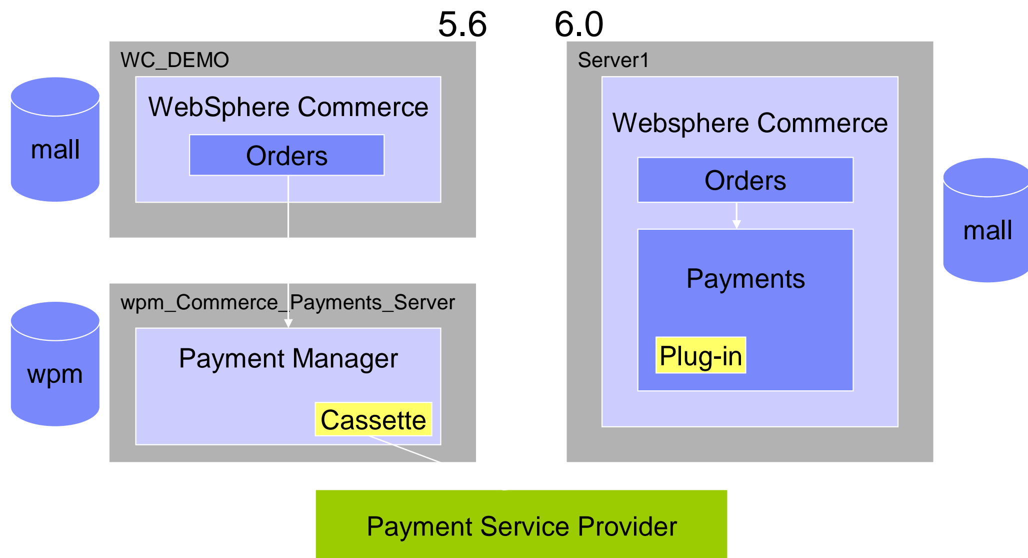


In version 6, the architecture is slightly different.

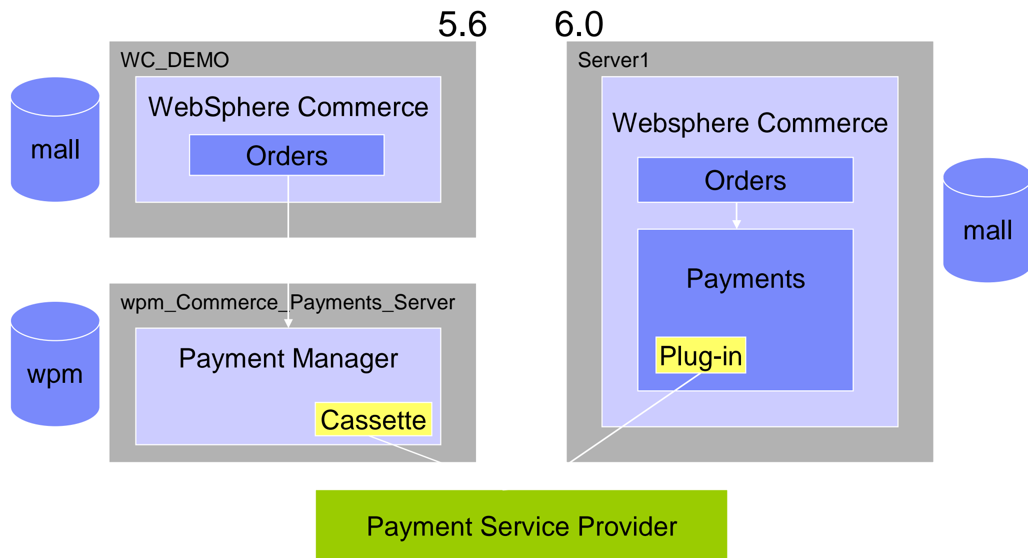
There is the same basic setup with the WebSphere Commerce application server and the orders component.



The difference now, is the addition of the Payments component into the WebSphere Commerce application. In essence, payments is no longer a separate application. It now resides entirely within the WebSphere Commerce Server and shares the same database.



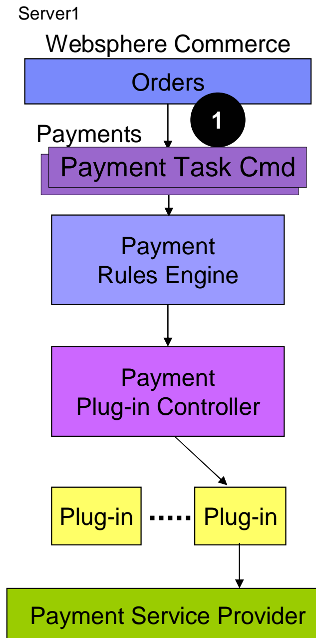
Requests flow from the Orders component directly to a Payments component without having to make a remote call.



At this point, the payments component takes over and uses a plug-in to communicate with the backend payment service provider.

Processing flow

1. OrderProcess (Submit) invokes a **payment task command** (PrimePayment)



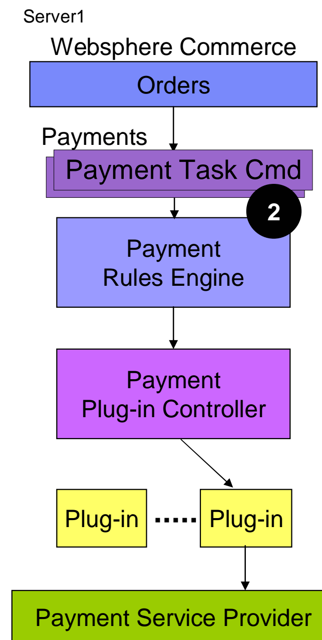
© 2008 IBM Corporation

This next section reviews the payment flow.

The first step in the payment flow begins when a WebSphere Commerce Controller command is invoked that requires some type of payment processing. For example, clicking submit on the shopping cart page calls the OrderProcess command. This in turn invokes the payments sub-system by calling the PrimePayment task command.

Processing flow

1. OrderProcess (Submit) invokes a **payment task command** (PrimePayment).
2. The **payment task command** (PrimePayment) calls the **Payment Rules Engine**.
3. The **Payment Rules Engine** determines the **Action** that needs to be performed (Approve).

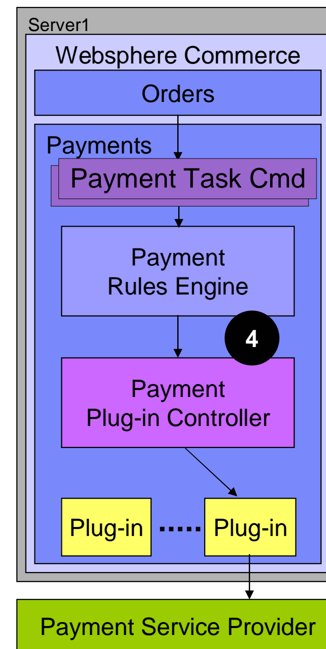


© 2008 IBM Corporation

In steps 2 and 3, the payment policy command calls the Payment Rules Engine component which determines which action needs to be performed.

Processing flow

1. OrderProcess (Submit) invokes a **payment task command** (PrimePayment).
2. The **payment task command** (PrimePayment) calls the **Payment Rules Engine**.
3. The **Payment Rules Engine** determines the **Action** that needs to be performed (Approve).
4. The **Action** is wrapped into an **Event,** which is passed to the **Payment Plug-in Controller**.

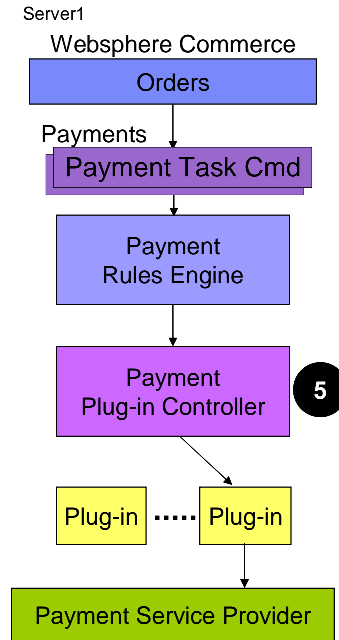


© 2008 IBM Corporation

In step 4, the action is wrapped into an event, which is passed to the Payment Plug-in Controller.

Processing flow

1. OrderProcess (Submit) invokes a **payment task command** (PrimePayment).
2. The **payment task command** (PrimePayment) calls the **Payment Rules Engine**.
3. The **Payment Rules Engine** determines the **Action** that needs to be performed (Approve).
4. The **Action** is wrapped into an **Event,** which is passed to the **Payment Plug-in Controller**.
5. The **Payment Plug-in Controller** determines the **Plug-in** to be used.

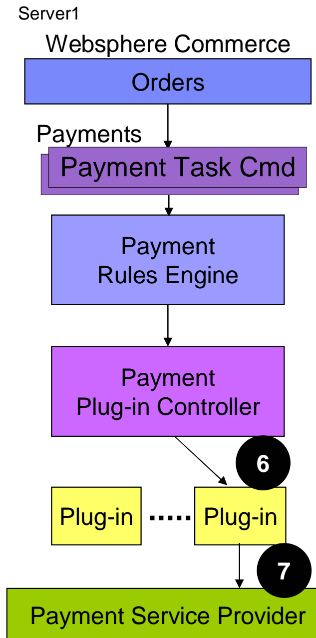


© 2008 IBM Corporation

In step 5, the payment plug-in controller determines which plug-in to invoke to perform the given request.

Processing flow

1. OrderProcess (Submit) invokes a **payment task command** (PrimePayment).
2. The **payment task command** (PrimePayment) calls the **Payment Rules Engine**.
3. The **Payment Rules Engine** determines the **Action** that needs to be performed (Approve).
4. The **Action** is wrapped into an **Event,** which is passed to the **Payment Plug-in Controller**.
5. The **Payment Plug-in Controller** determines the **Plug-in** to be used.
6. The **Action** is invoked against the **Plug-in**.
7. The **Plug-in** interacts with the Payment Service Provider.



© 2008 IBM Corporation

In step 6, the payment plug-in controller invokes the action against the plug-in, and the plug-in communicates with the backend payment service provider in step 7.

Payment method

- In the Order Summary page, the shopper gets to choose the payment method:

ADD PAYMENT METHOD

*Payment method:

VISA Credit Card

Pay later
VISA Credit Card
Check
Master Card Credit Card
Cash on delivery
American Express Credit Card
Bill me later

POLICY_ID	10002
POLICYNAME	**VISA**
STOREENT_ID	10001
PROPERTIES	attrPageName=StandardVisa&paymentConfigurationId=default&display=true&compatibleMode=false

- The payment method is associated with a payment policy. For example:

```
db2 "select policy_id, policyname, storeent_id, properties from
policy where storeent_id=10001 and policyname='VISA'"
```

© 2008 IBM Corporation

The remainder of this presentation will take you through a step by step example of payment processing.

First, take a look at the payment method.

This is where the payment processing typically begins.

When a shopper reaches the payment screen in their shopping flow, they are provided with a list of options on possible payment methods.

In this example, you see a drop down menu with options such as Visa and MasterCard.

Each of these payment methods are defined in the WebSphere Commerce database as a payment policy.

For this example, the VISA policy is defined in the drop-down list.

Note the values defined in the properties field.

These options are used to define how the payment method is used.

You can find more information on each of these options in the WebSphere Commerce Information Center.

Payment instruction

- A payment instruction is a payment method plus the details necessary to perform the payment action.
- Shopper adds payment instruction
  - PIAdd command runs
  - Persists the payment information in ORDPAYINFO
  - Creates a payment instruction in EDPPAYINST.

© 2008 IBM Corporation

Now that you have decided on the payment method, the next step is defining the payment instruction.

The payment instruction is a payment method plus any additional information necessary to perform the payment action.

For example, if you choose VISA card as your payment method, you also need to provide additional information such as the VISA card number and the expiration date.

Within the WebSphere Commerce server, when a shopper provides the payment instruction, the PIAdd command is invoked.

The PIAdd command is responsible for persisting the provided information into the ORDPAYINFO *(pronounced ORD-PAY-INFO)* table.

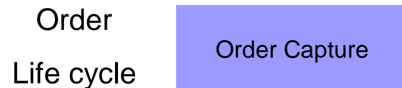
In addition, it creates a new record in the EDPPAYINST *(pronounced EDP-PAY-INST)* table describing the payment instruction.

Once the payment instruction has been provided, the order is submitted.

Business event

- An Order command, such as OrderProcess invokes a Payment Rules task command, such as PrimePaymentCmd.
- The task command raises a corresponding business event.

Order
Life cycle



Payment
Processing

© 2008 IBM Corporation

After the order is submitted, the payment component is initiated using Business Events.

Staying with the same Order Capture example, when an order is submitted, the OrderProcess command is invoked.

This calls a Payments Rule task command called PrimePayment which creates a corresponding PrimePayment Business Event.

The Business Event is used by the Payments Rule Engine as shown on the next slide.

Another thing to note here is that Order Capture is only one of several phases in the order life cycle.

Business event

- An Order command, such as OrderProcess invokes a Payment Rules task command, such as PrimePaymentCmd.
- The task command raises a corresponding business event.

Order Capture	ReleaseToFulfillment	Shipping
PrimePayment	ReservePayment	FinalizePayment

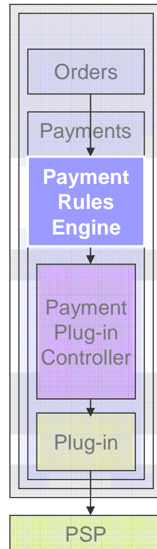
© 2008 IBM Corporation

Other phases include ReleaseToFulfillment and Shipping.

Similar to the Order Capture example, during execution of each of these phases, a corresponding business event is created.

The ReleaseToFulfillment phase creates a ReservePayment event, while a FinalizePayment event is created during shipping.

Payment rules engine



- The Payments Rules Engine determines which action needs to be taken based on the payment instruction/method and the business event.

Payment Instruction (**VISA + data**)
+ Business Event (**PrimePayment**)
Event (**Approve**)

1. Determine the Payment Action Rule
2. Determine the target state
3. Determine the actions
4. Create an event for the actions

© 2008 IBM Corporation

The business events are now consumed by the Payment Rules Engine.

The Payment Rules Engine is used to determine which actions need to be taken based on the business event that was raised and the corresponding payment instructions.

Continuing with the order capture example, you have a PrimePayment business event and the payment instructions for a Visa Card payment method.

The blue box in the diagram represents what the Payment Rules Engine does before deciding what actions to take.

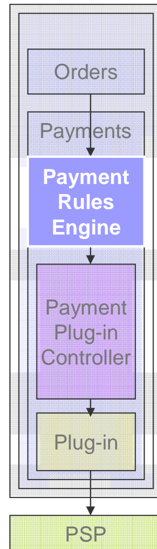
First, it determines the Payment Action Rule.

Using the rule, it determines the target state.

Once you know the target state, it determines what the necessary actions are to achieve it.

These actions are then wrapped in an event so that it can be passed onto the next component.

Payment rules engine



1. Determine the Payment Action Rule:

Payment Method (**VISA**)

Payment Configuration + Payment Action Rule
(**CreditCardOnline**) (**Early Approval**)

- xml/config/payments/edp/groups/**default**/PaymentMapping.xml

```
<Mapping paymentMethod="VISA"
   paymentConfiguration="CreditCardOnline"
      paymentActionRule="Early Approval"/>
```

© 2008 IBM Corporation

The first step in the Payment Rules Engine is to determine the Payment Action Rule.

The Payment Action Rule is determined by looking at the PaymentMapping.xml file located inside the WebSphere Commerce EAR.

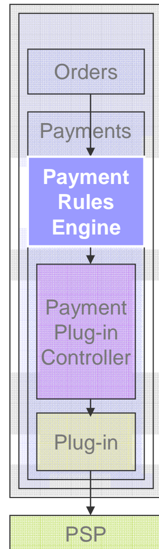
Based on the XML definition, it tells the rules engine what the payment configuration and payment action rule are.

In this example, the payment configuration used is CreditCardOnline, which defines how the payment method is associated to a payment backend system.

It also tells the rules engine to use the early approval payment action rule which defines the payment action behavior.

Next, you will see how these rules are used to determine the target state.

Payment rules engine



2. Determine the target state:

Payment Action Rule + Business Event

(**Early Approval**) (**PrimePayment**)

Target State

(**APPROVED**)

- xml/config/payments/edp/groups/default/PaymentRules.xml

```
<PaymentRule name="Early Approval">
    <PrimePaymentEvent targetState="APPROVED" />
    <ReservePaymentEvent targetState="APPROVED" />
    <FinalizePaymentEvent targetState="DEPOSITED"
  />
</PaymentRule>
```

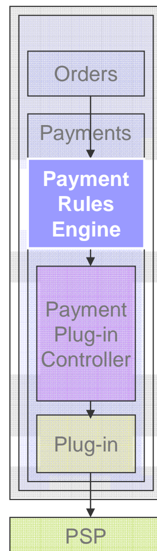
© 2008 IBM Corporation

To determine the target state, the Payment Rules Engine uses another configuration file called the PaymentRules.xml.

The PaymentRules.xml is used to map the payment rule that you determined in the previous step to the corresponding target state based on the incoming Business Event.

In this example, since you are receiving a PrimePayment business event, you can see that it maps to the APPROVED target state.

Payment rules engine



3. Determine the actions:

Payment Configuration (**CreditCardOnline**) + Target State (**APPROVED**) + Current State (**DNE**)

Action(**Approve**)

- xml/config/payments/edp/groups/default/**CreditCardOnline**/ CorePaymentActions.xml

```
<TargetApproved>
  <CurrentDNE>
       <Action name="Approve" amount="requested"
         target="new"/>
  </CurrentDNE>
</TargetApproved>
```


© 2008 IBM Corporation

The rules engine now uses another configuration file called the CorePaymentActions.xml.

Notice that there are multiple CorePaymentAction.xml files located within the WebSphere Commerce EAR.

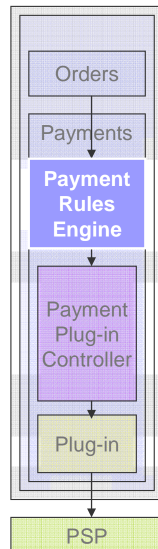
In order to select the right file, the rules engine uses the payment configuration identified in the previous step to locate the appropriate directory where the XML file is stored.

Using the XML file, you are now able to locate the target state and find the corresponding action based on the current state.

In this example, since you are submitting a new order, the current state is DNE, which stands for Does Not Exist.

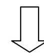
As you can see, to go from DNE state to Approved state, you need to call the Approve action.

Payment rules engine



Payment Instruction (**VISA + data**)
+ Business Event (**PrimePayment**)

1. Determine the Payment Action Rule
2. Determine the target state
3. Determine the actions
4. **Create an event for the actions**



Event (**Approve**)

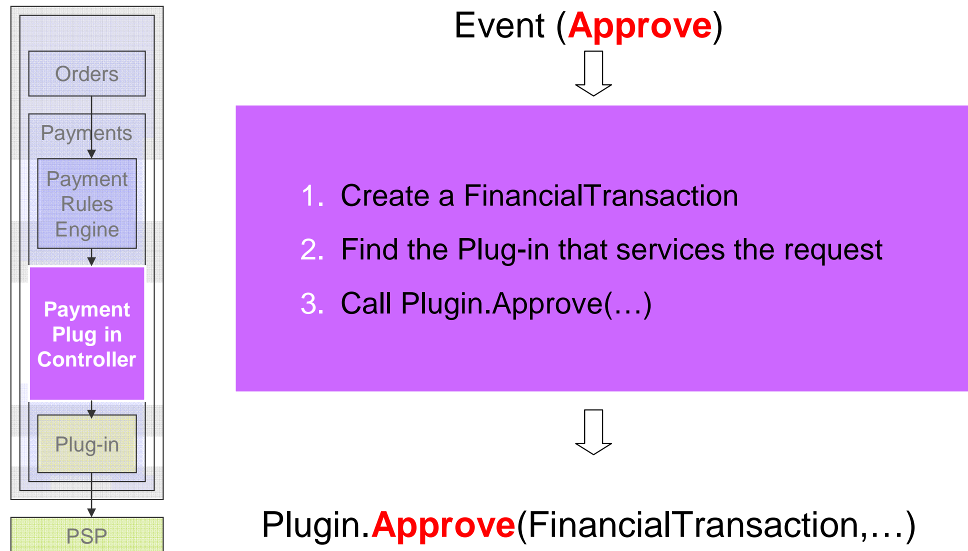
© 2008 IBM Corporation

For the last step in the Payment Rules Engine component, you need to create an event for the actions you have identified.

The reason for creating an event is to allow both synchronous and asynchronous flow.

This enables the Payment Rules Engine to decide whether it needs to wait for a response from the next component that is receiving the event or not.

In this example, you have created an Approve event.



The event is now passed to the Payment Plug-in Controller.

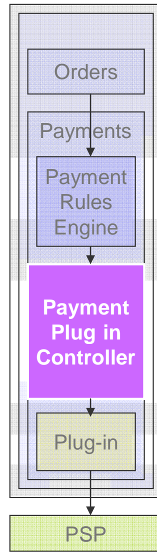
The payment plug-in controller is responsible for creating a financial transaction, finding the appropriate plug-in, and invoking the action against the plug-in.

The next slides cover how each of these steps work.

IBM Software Group

IBM

Payment plug-in controller



1. Create a FinancialTransaction

- A FinancialTransaction is used to track the progress of the Action as it is processed.
- It is populated with data associated to the Action and the Payment Instruction.
- The FinancialTransaction object contains attributes:
  - state: new
  - requested amount: 359.000
  - processed amount: 0.00000
  - reference number: …
  - response code: …
  - payment ID: 10001
  - …
- The transaction data is stored in the PPCPAYTRAN table

32

Payments technical overview

© 2008 IBM Corporation

The first thing that happens is the creation of a Financial Transaction object.

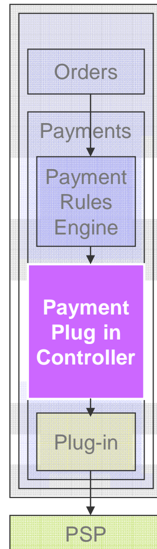
A Financial Transaction is an object that is used to track the progress of the actions as it is processed.

The financial transaction is populated with data associated with the requested Action and the corresponding payment instruction.

This example shows passing information that was collected earlier as part of the payment instruction such as the requested amount.

All of this is stored in the PPCPAYTRAN table.

Payment plug-in controller



2. Find the plug-in that services the request

- The plug-in is identified using the Payments System name from the Payment Instruction and a mapping file:

  Payment Method (**VISA**)

  Payment Configuration + Payment Action Rule
  (**CreditCardOnline**) (**Early Approval**)

- xml/config/payments/edp/groups/default/PaymentMapping.xml

```
<Mapping paymentMethod="VISA"
     paymentConfiguration="CreditCardOnline"
     paymentActionRule="Early Approval"/>
```


© 2008 IBM Corporation

Next, the Payment Plug-in Controller determines what the appropriate plug-in is to perform the request.

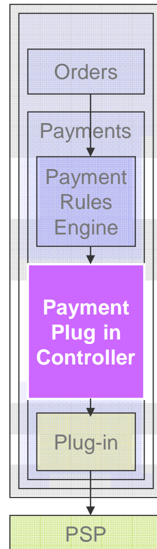
The first thing the controller looks at is the Payment Method that comes as part of the Payment Instruction.

In this case, the payment method is VISA.

Similar to the first step in the payment rules engine, it needs to determine the payment configuration that is described in the PaymentMapping.xml file.

In this example, the payment configuration is “CreditCardOnline”.

Payment plug-in controller



2. Find the plug-in that services the request (continued)

Payment Configuration (**CreditCardOnline**)

Payment System (**Paymentech**)

- xml/config/payments/edp/groups/default/PaymentMethodConfigurations.xml

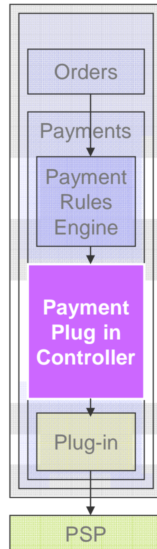
```
<PaymentMethodConfiguration
    name="CreditCardOnline"
    paymentSystemName="Paymentech" …/>
```


© 2008 IBM Corporation

The file PaymentMethodConfigurations.xml determines the payment system name based on the payment configuration.

Shown in this example, the payment system name for the CreditCardOnline configuration is Paymentech.

Payment plug-in controller



2. Find the Plug-in that services the request (cont'd)

Payment Configuration (**CreditCardOnline**)

Payment System (**Paymentech**)

- xml/config/payments/edp/groups/default/PaymentMethodConfigurations.xml

```
<PaymentMethodConfiguration
   name="CreditCardOnline"
   paymentSystemName="Paymentech" …/>
```

Payment System (**Paymentech**)

Plug-in (**PaymentechPlugin**)

- xml/config/payments/ppc/plugins/PaymentSystemPluginMapping.xml

```
<PaymentSystemName name="Paymentech" >
   <Mapping paymentConfigurationId="default"
   pluginName="PaymentechPlugin" >
```

© 2008 IBM Corporation

For the last step, you use the payment system name to identify the plug-in.

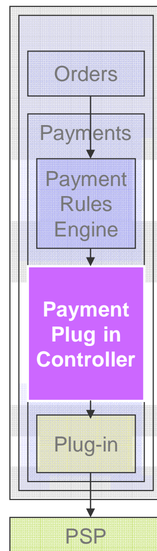
In order to identify the plug-in, you use a third file called PaymentSystemPluginMapping.xml.

As you can see from the XML snippet, the payment system name “Paymentech” maps to a plug-in called “PaymentechPlugin”.

IBM Software Group

IBM

Payment plug-in controller



3. Calls Plugin.Approve(…)

- Calls Plugin.Approve(…), in this case: PaymentechPlugin.Approve(…)

Event (**Approve**)

1. Create a FinancialTransaction
2. Find the Plug-in that services the request
3. **Call Plugin.Approve(…)**

Plugin.**Approve**(FinancialTransaction,…)

36

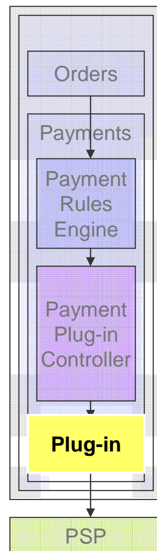
Payments technical overview

© 2008 IBM Corporation

Now that you have determined the appropriate plug-in, the payment plug-in controller is finally able to invoke the requested action on the plug-in.

For this example, the Approve action on the Paymentech plug-in is called, passing along the Financial Transaction object that was created.

Plug-in



Plugin.**Approve**(FinancialTransaction,…)

1. Retrieve data from the FinancialTransaction to be processed
2. Call backend Payment Service Provider (PSP) to perform the action
3. Populate FinancialTransaction with response data

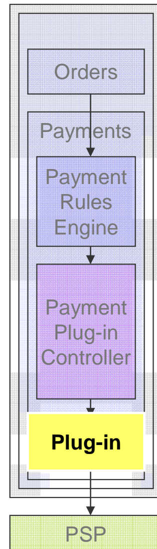
© 2008 IBM Corporation

Each plug-in contains specific logic to interact with a back-end payment service provider.

The plug-in retrieves data from the FinancialTransaction object, and uses this information to call the backend payment service provider to perform the required action.

Then, based on the response, it populates the FinancialTransaction object and sends it back to the payment plug-in controller to persist the data.

Plug-in



- The plug-in implements API (com.ibm.commerce.payments.plugin.QueryablePlugin)
- This API has methods that map to the actions to be performed

```
public FinancialTransaction approve(...
public FinancialTransaction deposit(...
public FinancialTransaction credit(...
public FinancialTransaction
approveAndDeposit(...
public FinancialTransaction reverseApproval(...
public FinancialTransaction reverseCredit(...
public FinancialTransaction reverseDeposit(...
...
```

- The implementation of these methods communicates to the backend (PSP) using their backend specific APIs

© 2008 IBM Corporation

Here is a closer look at how the plug-in works.

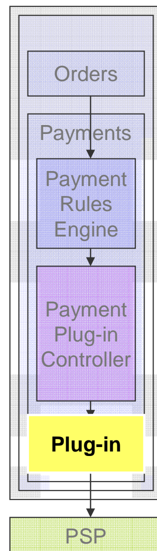
First of all, the plug-in implements an API called QueryablePlugin.

This API defines the interfaces for the methods that map to each action that needs to be performed.

For example, here are some actions, including approve, deposit, and credit.

The plug-in implements the logic for each one of these methods, including protocol specific instructions to communicate with the backend payment service provider.

Plug-in



- IBM Supported payment plug-ins
  - SimpleOffline payment plug-in
    - Does not directly communicate with a payment back-end system.
    - Records events that have already happened outside of WebSphere Commerce. Transactions are recorded and maintained in the WebSphere Commerce database.
  - Line of credit (LOC) payment plug-in
    - The LOC plug-in mimics the function of an actual line of credit. It does not connect to an account receivable system
  - WCPayments Cassette plug-in
    - For use with the traditional WebSphere Commerce Payments.
    - Acts as a bridge to the traditional payment framework (Payment Manager) and the use of payment cassettes.
  - Paymentech plug-in
    - Added as a feature in WebSphere Commerce Feature Pack 2
    - Communicates directly with the Paymentech gateway through the plug-in

© 2008 IBM Corporation

Here is a list of all the standard plug-ins that are supported for WebSphere Commerce.

The SimpleOffline plug-in is the most basic plug-in because it does not communicate with the backend payment service provider.

It essentially retains the payment information for business users to then take that information to perform the payment manually.

It allows the business user to update the state accordingly and persist it into the database.

Line of Credit plug-in also does not communicate with a backend payment system. It is similar to the SimpleOffline plug-in, but performs line of credit payment processing.

The WCPayments Cassette plug-in, is used as a bridge to communicate with the old 5.6 payment manager server.

This allows you to use the new Payment plug-in architecture to communicate with an existing payment manager server which uses a cassette to perform transactions with the backend payment service provider.

You might choose this approach if you want to keep using your existing custom cassette written for payment management until that cassette can be converted to a payment plug-in.

The Paymentech plug-in is the only plug-in that is not included as part of the base installation. It is part of Feature Pack 2.

This plug-in allows your system to communicate directly with the Paymentech gateway.

Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send e-mail feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_wcs60_PaymentsTechnicalOverview.ppt

This module is also available in PDF format at: ../wcs60_PaymentsTechnicalOverview.pdf

© 2008 IBM Corporation

You can help improve the quality of IBM Education Assistant content by providing feedback.

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 WebSphere

A current list of other IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

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 or changes in the products or programs 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 (for example, 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 2008. 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.