CHAPTER 3

DESIGN OF SYSTEM ARCHITECTURE

3.1 System Overview

The design of the service will be along the line of a buffering service which will buffer expected message when the receiving flow is not ready. The service will reside on each of the participating flows so that it is also applicable on the cross-organizational level [9] and will serve as a gateway, redirect outgoing messages to another gateway where the remote service is located. The overview of the service will be as depicted:

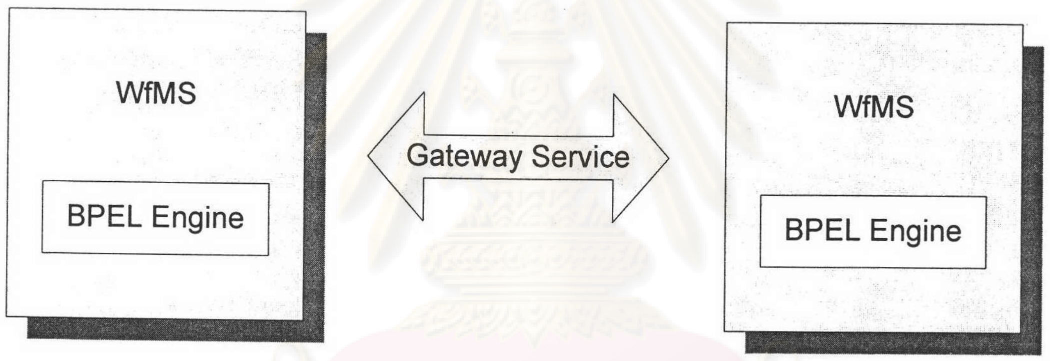

Figure 3-1 The overview of the proposed design of the Gateway Service

There are some aspects that need to be considered which are:

1. Since the service acts as a buffer for a BPEL web service which communicates through SOAP, the service itself could easily be a web service as well.
2. The procedure of discovering desired services and establishing contracts will not be discussed.
3. For the purpose of interoperability the service needs to be generic in nature and no prior knowledge of the relating flow is needed i.e. the service is applicable to any BPEL flow.
4. Each participant has a gateway responsible for its incoming message in the manner of a de-centralized system. As a centralized control is usually followed

by high cost of communication and reduced autonomy of the participating flows [9].

5. The Gateway Service buffers message only after some kind of validation has been made – random or meaningless message should be discarded.
6. Each outgoing message will be sent through a gateway on its side which will redirect the message to the gateway corresponding to the destination service location.
7. Thus, in short, a gateway service will be responsible for two main tasks which are
    a. Redirecting outgoing messages designated for a particular service to the corresponding gateway.
    b. Buffering valid incoming message for a service when its `<receive>` activity is not ready in order for the service to fetch the buffered message later

Consider a fundamental case of two collaborating flow with a Gateway Service on each side, the simplest case of the collaboration would be one of the flow sending a one-way message to the other. Let Flow A be the flow on the sending side, and Flow B be the flow on the receiving side. In the same manner, let the Gateway Service on each side be called Gateway A and Gateway B. The procedure will be as followed:

1. Flow A sends a message out to Gateway A, expressing the need for the message to be redirected to Flow B.
2. Gateway A looks up the location of Flow B and finds that the message must be passed on to Gateway B for it being the corresponding Gateway.
3. Meanwhile, Flow B knows that a message is to be expected from Flow A and register this fact to the Gateway B.
4. Once the message arrives at Gateway B, the Gateway validates the message. If the message is valid.

The first four steps of the process are depicted in Figure 3-2.

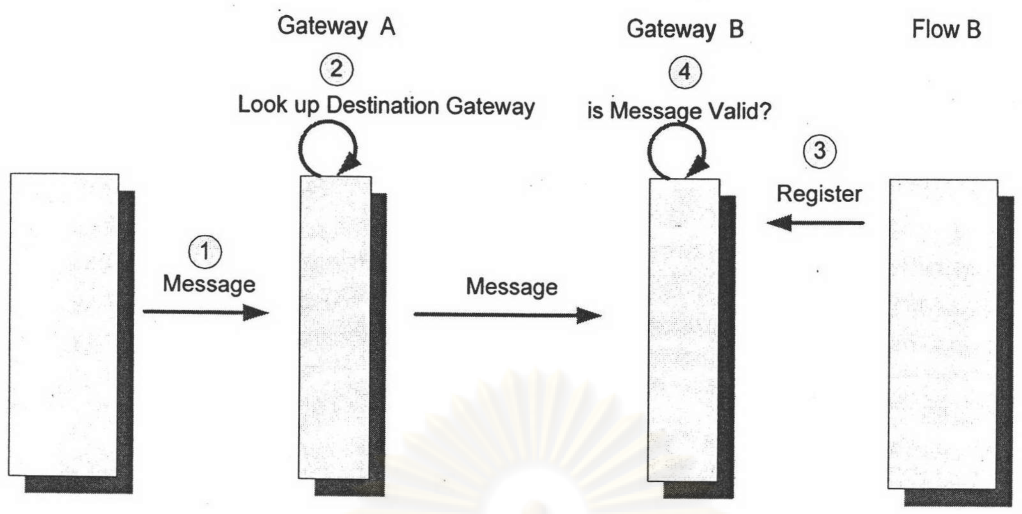

Figure 3-2 The initial processes

5. If a message is valid, there are three possible scenarios that could happen
    a. Case A: The Flow B is not yet ready to receive the incoming message so the process from this point would be (Figure 3-3):
        i. Gateway B buffers the message and waits.
        ii. Once Flow B is ready it notify that to Gateway B
        iii. Gateway B sends the buffered message to Gateway B.
    b. Case B: The Flow B has already notified that it is ready to receive the incoming message and so the process from this point would be that the Gateway passes message on to Flow B; no buffering is needed (Figure 3-4).
    c. Case C: The Flow B has not been instantiated at this point but an incoming message creates a new instance. Thus no buffering is needed in this case either; the message passed on and a new instance of Flow B is created. It is important to note that prior registration is not needed (Figure 3-3).

Note that procedure 1 - 4 are the same in three cases, with an exception of Case C, where no advance registration is needed i.e. process 3 is missing.

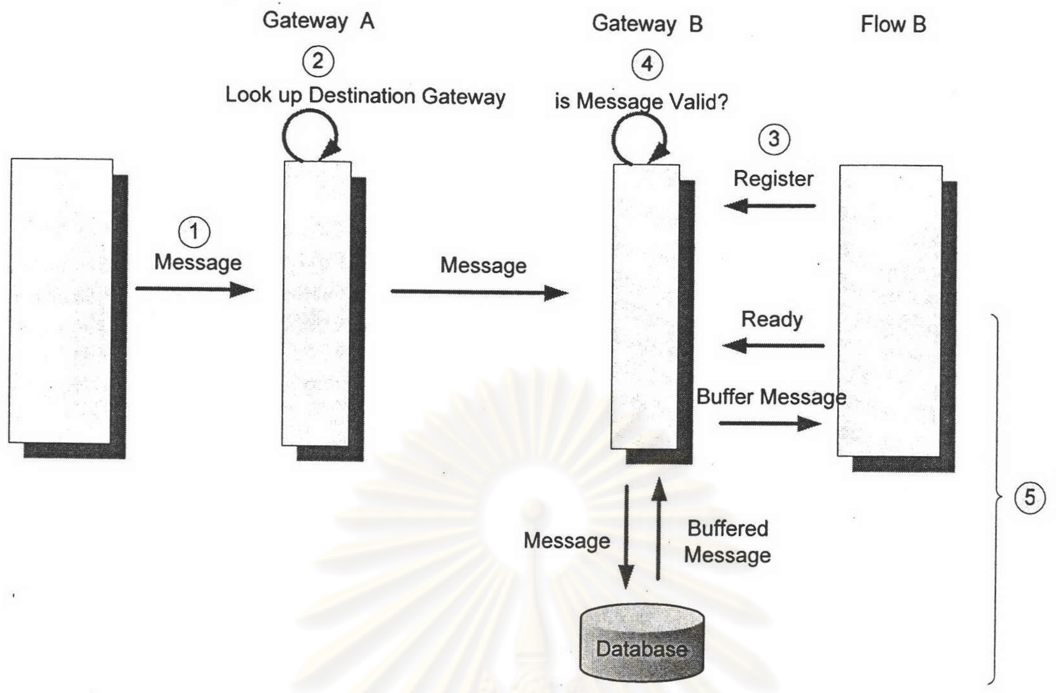

Figure 3-3 Case A where the message arrives before the receiving flow is ready

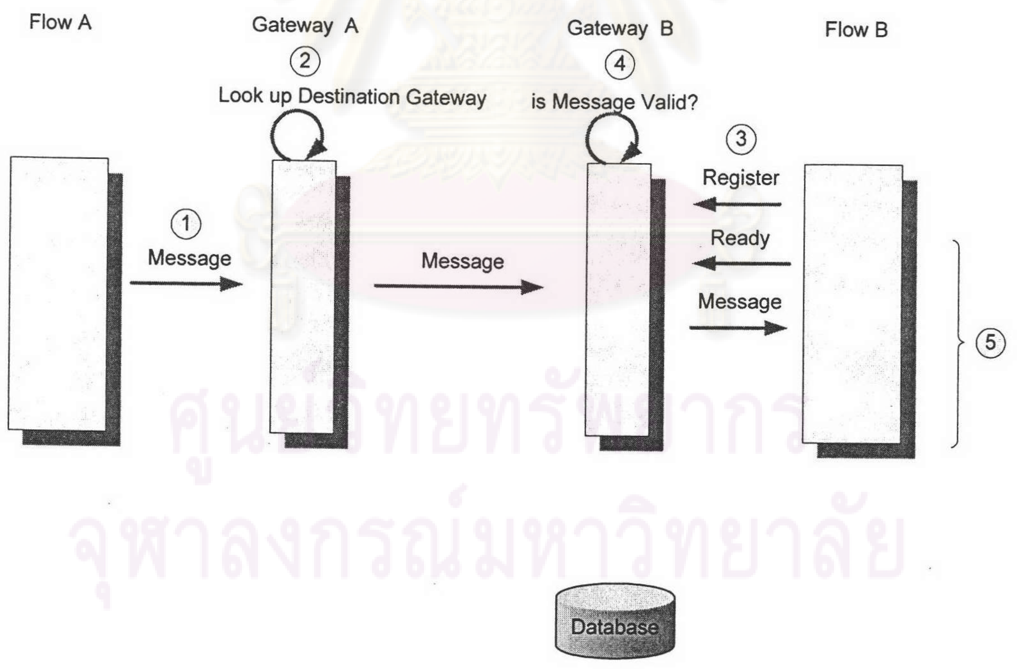

Figure 3-4 Case B where the message arrives after the receiving flow enters the ready state

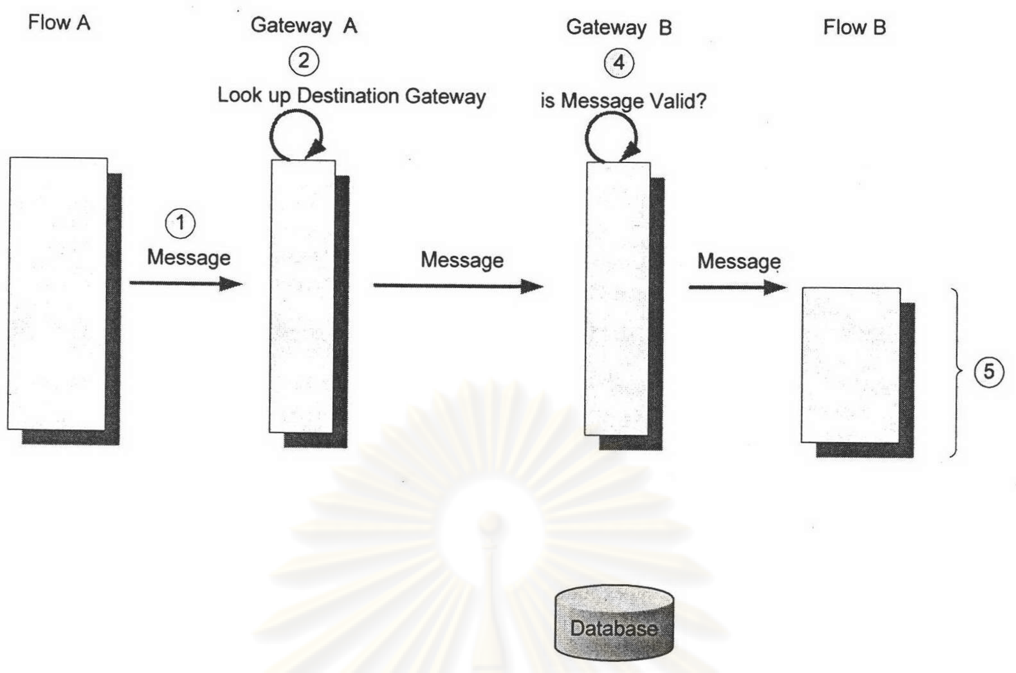

Figure 3-5 Case C where the incoming message creates a new instance of the receiving flow

3.2 Message Format

Several XML message formats relating to the Gateway Service, which is communicating on SOAP, have been defined. (Figure 3-6). Note: all the attributes are of the type String, unless specified otherwise.

1. The most important message format is called "GatewayMessage", having two main parts which are the message header and the message body.
    a. The message header (MHeader) is the gateway specific message containing the information regarding where to redirect messages and correlation data. The message header includes the following fields:
        i. jointFlowID – Each participant must agree on a "jointFlowID", which is unique across the participating organization for each flow connecting manner, prior to the execution of the flow. Assuming that only the participants in the collaboration can provide the correct jointFlowID, this is used to verify the validity of the incoming messages. For example if two collaboration have the same participating BPEL flows which however, are connected in different manner, two jointFlowID's

must be issued to be able to tell them apart . The real content of the jointFlowID could range from something looking as simple as “001@org1@org2@org3” to any form of a more elaborated authentication key with secured identification method.

ii. instanceID – As the “jointFlowID” is unique upon a manner of flow connecting, another id used to distinguish each instance must also be included, so that the gateway can deliver the message to the correct instance. This could be any correlation data such as purchase order or invoice number.

iii. callbackURL – the callback address of the service whose message is being sent out, this is used for the remote service to callback asynchronously.

iv. serviceURL – The location of the remote service.

v. serviceNameSpace – The namespace of the service

vi. serviceName – The name of the service for the message to be directed to

vii. portType – the portType as defined in WSDL of the remote service

viii. soapAction – the soapAction which is also defined in WSDL

ix. entryPointID – For a flow that branches and have multiple entry points for incoming message an ID or operation name is necessarily to be included in the message headed

b. In order for the gateway to be generic the message body (Mbody) is transmitted as plain text. However, the content of the text could be typed XML messages and thus, casting of XML typed data to text and vice versa is required as needed

Note: As the message body is transmitted as plain text, the correlation sets in the message data as specified in BPEL specification will not work. An array of strings could be added in to the header and serves as a correlation sets if necessary.

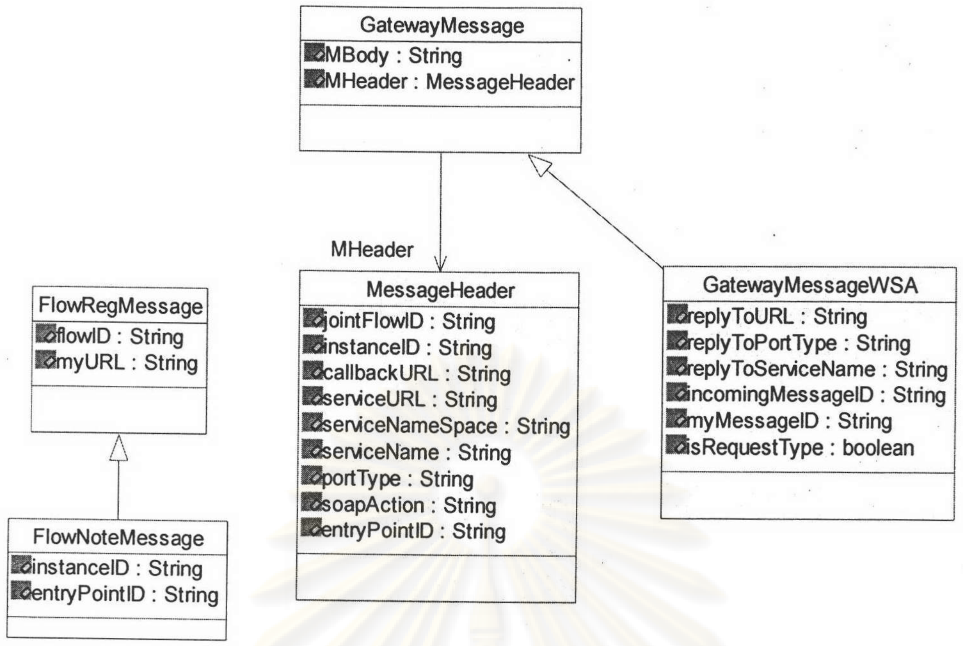

Figure 3-6 The class diagrams of the main message types

2. Another message format is called FlowRegMessage, this message is used by the receiving flow to register the FlowID. The class has two attributes which are:
    a. flowID, this is the alias for jointFlowID
    b. myURL is the URL of the service where the receiving flow is located
3. Another message format is called FlowNoteMessage, which is a subclass of FlowRegMessage, this message is used by the receiving flow to notify the GatewayService when it enters the ready state. The class has two additional attributes which are:
    a. instanceID, which is the same as the instanceID of the GatewayMessage
    b. entryPointID, this is also the same as the entryPointID in the class GatewayMessage
4. The last message format is GatewayMessageWSA. This is a subclass of GatewayMessage. The WSA abbreviation stands for WS-Addressing. This is

the message format that is stored in the database. In additional to the inherited attributes, the class has the following attributes:

a. replyToURL – this is extracted from the WS-Addressing field bearing the same name, in the header in the SOAP message. This tells the BPEL process where to send a reply message to. This also can be regard as the origin of the message.

b. replyToPortType – this is also from the WS-Addressing, in the header in the SOAP message. This tells the BPEL process the WSDL portType of the service to send a reply message to.

c. replyToServiceName – in the same manner, this is the WSDL service to reply to.

d. incomingMessageID – the ID that the incoming message bears.

e. myMessageID – the message ID which the gateway generates for the message that it receives.

f. isRequestType – the boolean value that tell whether this message is a request or a response message.

3.3 System Design

The system has been designed upon the basis that enactment environment correlates mainly using WS-Addressing. Considering the possible cases mention previously, the design focuses mainly on the first 2 cases, where the flow on the sending side knows in advance that there is a particular instance waiting for a message on the receiving side.

There are two main classes: the principle GatewayService class and the class GatewayDB which takes care of the database related tasks.

3.3.1 GatewayService Class

This class performs most of the skeleton tasks, carrying out most of the main logic in framework.

1. Attribute

a. guid :VMID – the virtual machine id which is unique across the net.

b. MY_ADDRESS :String – the location of the Gateway Service

2. Methods

a. initiate – this method gets called from a collaborating flow when there is the need to send a message out to a remote service

b. delegatingOperationRequest – This method gets called from the initiate method. It basically looks up the address of the remote Gateway where the remote flow resides (**queryRemoteGatewayLocation**), generates a message id, registers the outgoing message id and replyTo information for the callback purpose.

c. queryRemoteGatewayLocation – this method returns the URL of a Gateway Service when given the URL of a flow.

d. releaseMessage – this method release the message to a flow when it is ready to receive.

e. releaseMessageVector – given a array (or Vector) of buffered messages this method release the message one by one (releaseMessage).

f. gatewayInboundRequest – this method is called from a peer Gateway Service to pass on a message or buffer it, depending on the condition. This method connects to the database to check if the incoming message has a valid jointFlowID. It processes the incoming message with valid jointFlowID and throw false otherwise.

g. gatewayInboundResponse – this method is called from a peer Gateway Service in order to pass on a response message or buffer it. In the same manner as the gatewayInboundRequest method, a message with invalid jointFlowID will result in false being thrown.

h. notifyReadyState – this method is called from a flow when it is ready to receive an incoming message

i. onResponse – this method is called from a flow when there is a need to reply back with a response message

j. registerFlowID – this method is called from a flow telling the gateway to buffer any incoming message bearing a given jointFlowID.

k. unregisterFlowID this method is called from a flow telling that there is no need to buffer anymore message with a certain jointFlowID.

3.3.2 GatewayDB Class

This class is an auxiliary class whose tasks are database related tasks instead of the GatewayService having to connect to the database directly.

1. Attribute

The Class has 4 static attributes whose purpose is for connecting and authenticating with the DBMS. The names and types are as followed:

a. DB_URL	:String
b. DB_NAME	:String
c. DB_PASSWD	:String
d. DB_USER	:String

2. Methods

The class has the following method:

a. getConnection – this method initialize connection to the database prior to any query or update could be made

b. queryRemoteGatewayLocation – this method looks for the location of the Gateway Service when given the URL of a given service

c. registerOutgoingMessage – this method put the incoming message id, and other WS-Addressing “replyTo” information along with the message id that is generated by Gateway Service

d. registerFlowID – this method is called when a flow participating in the collaboration tells the GatewayService that the message bearing such jointFlowID is being expected.

e. isFlowIDValid – this method checks if a given jointFlowID is valid. In case of an invalid jointFlowID, an exception is thrown and is propagating back to the flow where the message is originated.

f. bufferIncomingMessage – this method puts the incoming message onto the database.

g. readyState – this method update the database when a notification, indicating a flow is ready to receive incoming message, comes in.

h. getOutgoingMessage – this method fetches the buffers messages from the database when ready to be sent out.

i. queryRegisteredMessageID – this method perform an auxiliary function, given flowID, instanceID and serviceURL, it finds the messageID that matches the criteria.

j. isReadyToReceive – checks if a flow is ready to receive.

k. queryAddressforPassing – this functions looks for the query the replyTo address that comes with the message notify that the flow is ready for sometimes the flow's URL is not exactly the same as the replyTo address.

l. getCallBackHeader – retrieve the replyTo information needed in order to call back.

3.3.3 The order of execution

Again, consider the case of two collaborating flows with a Gateway Service on each side. In this case of the collaboration, one of the flows sends a message to the other and gets a response callback. Let Flow A be the flow on the sending side, and Flow B be the flow on the receiving side. In the same manner, let the Gateway Service on each side be call Gateway A and Gateway B. The gateway service has interfaces as depicted in the following sequence diagram (Figure3-7):

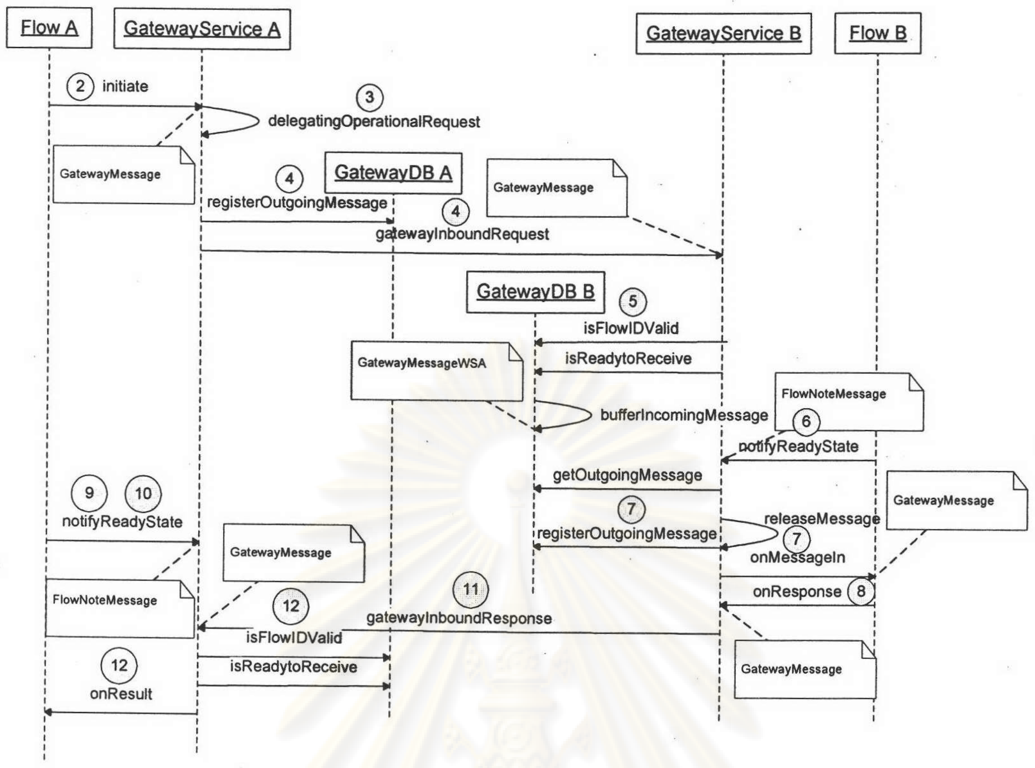

Figure 3-7 Sequence Diagram of the Implementation

Also depicted in Figure 3-7 is the message format being sent in between the flow, the GatewayService and the GatewayDB. The description of the service is as followed:

1. Prior to any cross-flow collaboration could be established, each participant must contact each other and agree on a flowID, and then registers it to its own gateway **(registerFlowID)** to ensure that any incoming message is being expected by a corresponding party. This process does not belong to any instance of a flow but must be done separately for many collaboration instances can occur under one jointFlowID.
2. A participant *(Flow A)* sends **(initiate)** a GatewayMessage, in the format described previously, to the gateway service on its side *(Gateway A)* expressing the need to contact another service *(Flow B).*

3. The *Gateway A* take care of the incoming message from Flow A (**delegatingOperationRequest**) by looks up the serviceURL in its database and sees that *Flow B* is behind *Gateway B*.
4. The Gateway generates a messageID for the outgoing message for callback purpose, and keeps records of the outgoing messageID (**registerOutgoingMessage**) along with the original messageID and redirects the message accordingly (**gatewayInboundRequest**).
5. *Gateway B* connects to the database and finds that the message bearing such flowID is being expected (**isFlowIDValid**) and check to see if Flow B is ready to receive (**isReadyToReceive**), in this case *Flow B* has not notified that it is ready yet so the message is buffered.
6. *Flow B* executes its business logic until it reaches the point where it is ready to receive an incoming message, then notifies the *Gateway B* (**notifyReadyState**) that it is ready to receive an incoming message.
7. Upon receiving the notification from *Flow B*, *Gateway B* looks up the buffered message, (**getOutgoingMessage**) generates a messageID, records it (**registerOutgoingMessage**), and forwards it to the *Flow B* (**onMessageIn**).
8. *Flow B* processes the incoming message and generates a result and needs to response to *Flow A*. So it sends out a GatewayMessage to the *Gateway B* (**onResponse**).
9. Meanwhile, *Flow A* too has come to the point where it is ready to receive an incoming message through its `<receive>` activity, so it sends out a notification to Gateway A (**notifyReadyState**).
10. Upon receiving the notification from *Flow A, Gateway A* search the message queue and finds no message waiting for *Flow A*.
11. *Gateway B* uses the messageID to finds the original of the message and finds that it has to be directed to *Gateway A*. (**gatewayInboundResponse**)
12. With the arrival of the response message, *Gateway A* checks (**isFlowIDValid**) and realizes that this message is being expected by *Flow A* which is also in the ready state (**isReadyToReceive**), so there is no need for buffering and the message is sent to *Flow A* (**onResult**). *Flow A* processes the rest of the

business logic accordingly. This completes the cycle of the business collaboration.

13. The jointFlowID can now be unregistered (**unregisterFlowID**) if no more in coming message in this manner of collaboration is expected.

From the sample flow described above, we can see that the procedure in receiving incoming message in step 5 – 7 on *Flow B* differs from that in step 12 on *Flow A*. The former case is an exceptional behavior of the message sending - receiving pattern – the receiving flow (*Flow B*) is not ready when the incoming message arrives at the Gateway Service. The case in step 12 however, is a normal behavior where the message arrives at the time where the receiving flow (*Flow A*) is ready. More details in exceptional cases are discussed in the next section.

3.4 Exceptional Cases

As the regular behaviors of the system have been discussed previously, it is also important to anticipate the exceptional behaviors of the Gateway Service and also to define what compensation procedures are to be carried out when such incidents occur.

1. **Method**: registerFlowID – In this method, the Gateway Service checks for duplicates entry of the jointFlowID and myURL. The new entry to be register can have the same value of either one of the attribute present in the database, but not both.

   Condition: Duplication occurs.

   Handling: The new entry will not be registered and an exception is thrown back to the BPEL flow where the message is originated as a notifying method that the message does not go through.

2. **Method**: delegatingOperationalRequest – This method looks for the Gateway Service URL using the destination service URL as a key.

   Condition: URL not found.

   Handling: A exception is thrown back the BPEL flow that sends out the outgoing message.

3. **Method**: delegatingOperationalRequest – The method calls the method gatewayInboundRequest at remote Gateway Service where the desitination service is located using the URL that has been looked up.

   Condition: False URL as a Gateway Service address

   Handling: An exception is also thrown back to the BPEL flow with the false message.

4. **Method**: gatewayInboundRequest – The method checks if the incoming message has a valid jointFlowID.

   Condition: invalid jointFlowID

   Handling: An exception is also thrown back to the Gateway Service that passed in this message which propagates the exception thrown to the remote BPEL flow where the message is originated.

5. **Method**: notifyReadyState – The method checks the jointFlowID with the database and also registers that the receiving flow is ready using the instanceID.

   Condition: invalid jointFlowID

   Handling: An exception is thrown back to the flow attempting to register its ready state.

6. **Method**: gatewayInboundResponse – The method checks if the incoming message has a valid jointFlowID.

   Condition: invalid jointFlowID

   Handling: An exception is thrown back to the Gateway Service passing in the response message which propagates the exception to the flow where the response message is originated.