

Agent Messaging

-----SENG609.22 Tutorial 2

Dong Liu

Abstract: The concepts of messaging and messaging system are presented. The basic components of messaging system are reviewed. The criteria and options of application integration are discussed. We can find that messaging is a good strategy for application integration based on those criteria.

1. Introduction

This topic is not on the list of tutorials of this course, but it is tightly related with agent-based system. Communication is a key feature of agents, and messaging is a specific method of communication. Messaging is a technology that enables high-speed, *asynchronous*, program-to-program communication with reliable delivery. Messaging capabilities are typically provided by a separate software system called a messaging system or message-oriented middleware. A messaging system manages messaging the way a database system manages data persistence. The messaging system coordinates and manages the sending and receiving of messages. Messaging system is discussed in section 2 of this tutorial.

Messaging and messaging system are applied in the cases that there are requirements about remote communication, platform/language integration, asynchronous communication, disconnected operation, thread management. Messaging has been applied in web services, and is a good option for agent-based system as well. Messaging is a method for enterprise integration that is discussed in section 3.

2. Messaging system

The basic components of messaging system include messages, message channels, message endpoints, and more specially, message filters, and message routers.

2.1 Message

A Message is an atomic packet of data that can be transmitted on a channel. Thus to transmit data, an application must break the data into one or more packets, wrap each packet as a message, and then send the message on a channel. Likewise, a receiver application receives a message and must extract the data from the message to process it. The message system will try repeatedly to deliver the message (e.g., transmit it from the sender to the receiver) until it succeeds. A message can usually be decomposed to a head and a body.

Normally, a message is transmitted in five steps:

1. Create. The sender creates the message and populates it with data.
2. Send. The sender adds the message to a channel.
3. Deliver. The messaging system moves the message from the sender's computer to the receiver's computer, making it available to the receiver.

4. Receive. The receiver reads the message from the channel.
 5. Process. The receiver extracts the data from the message.
- The steps are shown in Figure 1. If a message needs to be persistent, the transmission steps are different. A persistent message in JMS is shown in Figure 2.

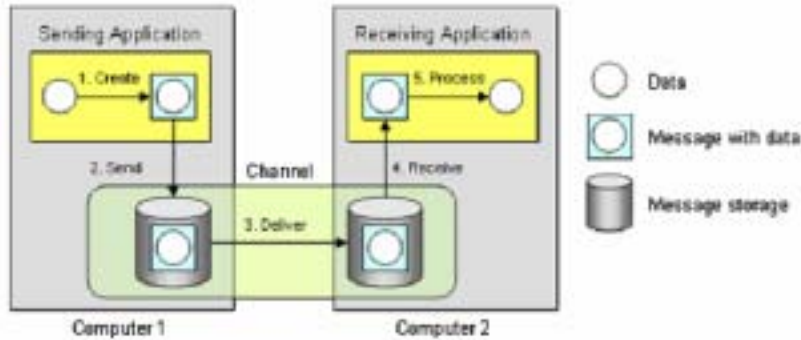


Figure 1. Revolution brought by XML [2]

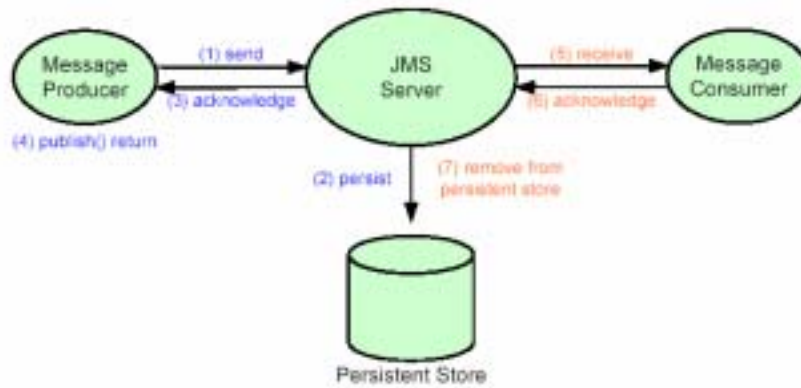


Figure 2. Transmission steps of a persistent message

2.2 Message Channel

Messaging applications transmit data through a Message Channel, a virtual pipe that connects a sender to a receiver. Channels are logical addresses in the messaging system; how they're actually implemented depends on the messaging system product and its implementation.

There are two different kinds of message channels, Point-to-Point Channels and Publish-Subscribe Channels. These two kinds of channels in JMS are shown in Figure 1. Applications may need multiple channels for transmitting different types of information and transmitting the same information to lots of other applications. Each channel requires memory to represent the messages; persistent channels require disk space as well. Even if an enterprise system has unlimited memory and disk space, any messaging system implementation usually has some hard or practical limit to how many channels it can service consistently.

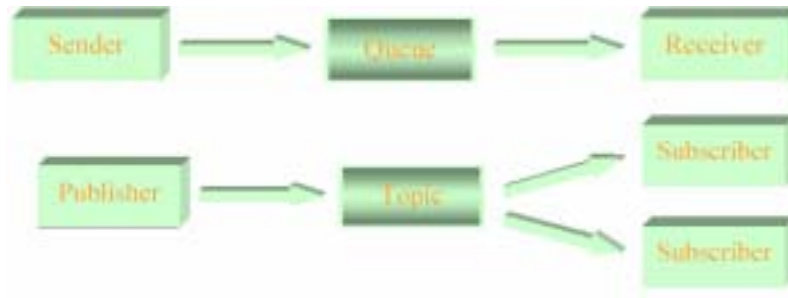


Figure 3. Point-to-Point Channels and Publish-Subscriber Channels

2.3 Message Endpoint

A Message Endpoint is a client of the messaging system that the application can then use to send or receive messages. Message Endpoint code is custom to both the application and the messaging system's client API. The rest of the application knows little about message formats, messaging channels, or any of the other details of communicating with other applications via messaging.

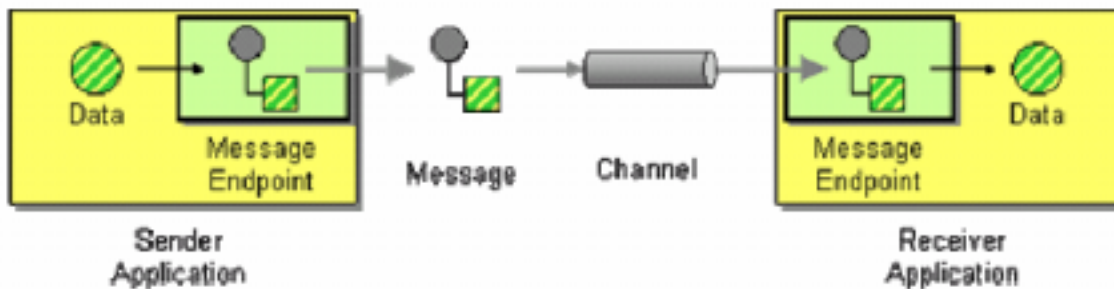


Figure 4. Message endpoints [2]

The Message Endpoint encapsulates the messaging system from the rest of the application, and customizes a general messaging API for a specific application and task. If an application using a particular messaging API were to switch to another, developers would have to rewrite the message endpoint code, but the rest of the application should remain the same. A Message Endpoint can be used to send messages or receive them, but one instance does not do both. An endpoint is channel-specific, so a single application would use multiple endpoints to interface with multiple channels. An application may use more than one endpoint to interface to a single channel, usually to support multiple concurrent threads.

2.4 Filter

The filter are a kind of component that recognizes different received messages and trigger the system to behave differently according to different messages. In many enterprise integration scenarios, a single event triggers a sequence of processing steps, each performing a specific function. Just like the system being developed by Tong Chen, the agent need to perform 5 tasks, but receive only one message. A filter can help the trigger the right task or tasks based on that message. A sample of filter is shown in Figure 5. The incoming message is clean step by step and the outcome message is ready for processing.

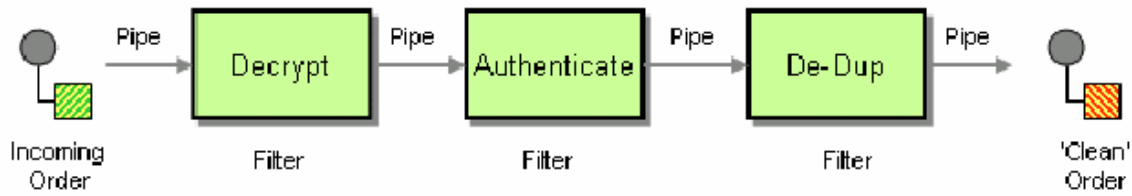


Figure 5. Message filters [2]

2.5 Message Router

A Message Router consumes a message from one message channel and republishes it to a different message channel depending on a set of conditions. A message router comprises at least a message filter that supports the switch decisions.

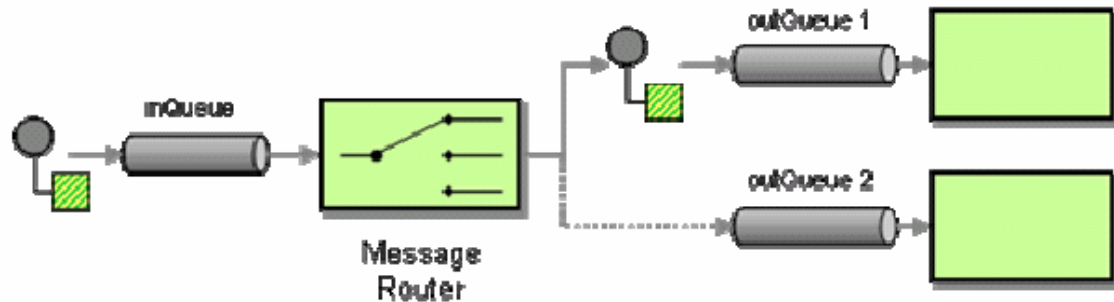


Figure 6. Message router [2]

3. Application Integration

Since messaging is supposed to be applied in application integration, the criteria and options of application integration are discussed in this section. We can find that they are very similar to those of agent-based systems.

3.1 Application Integration Criteria

We can summarize the criteria as follows:

Application coupling. The integrated applications should minimize their dependencies on each other so that each can evolve without causing problems for the others. Tightly coupled applications make numerous assumptions about how the other applications work; when the applications change and break those assumptions, the integration breaks. The interface for integrating applications should be specific enough to implement useful functionality, but general enough to allow that implementation to change as needed.

Integration simplicity. When integrating an application into an enterprise, developers should strive to minimize changing the application and minimize the amount of integration code needed. Yet changes and new code will usually be necessary to provide good integration functionality, and the approaches with the least impact on the application may not provide the best integration into the enterprise.

Data format. Integrated applications must agree on the format of the data they exchange, or must have an intermediate translator to unify applications that insist on different data formats.

Functionality. Integrated applications may share functionality such that each application can invoke the functionality in the others.

Asynchronicity. The source application may simply make shared data available or log a request for a procedure call, but then go on to other work confident that the remote application will act sometime later.

All these criteria are all valid for the agent-based system.

3.2 Application Integration Options

Generally, there are 4 strategies for application integration.

File Transfer. Each application produces files of shared data for others to consume, and consumes files that others have produced.

Shared Database. The applications store the data they wish to share in a common database.

Remote Procedure Invocation. Each application exposes some of its procedures so that they can be invoked remotely, and applications invoke those to run behavior and exchange data.

Messaging. Each application connects to a common messaging system and exchanges data and invokes behavior using messages.

Which strategy should be applied depends on the scenarios of application integration. Messaging is a good decision for agent-based system based on the criteria discussed in section 3.1.

4. Conclusion

Messaging will be a primary strategy for application integration with the development of web services. It will be the preferred communication solution for agent-based system. In such cases, messages are the key artefacts in system analysis and design. There are no good tools in UML to model the message analysis and design, so maybe we need to develop a new method based on data flow modeling for message-centric agent-based system and integrated applications.

References

- [1] Frank P. Coyle, XML, Web Services, and the Data Revolution. Addison Wesley, 2002.
- [2] Gregor Hohpe, Bobby Woolf, Enterprise Integration Patterns : Designing, Building, and Deploying Messaging Solutions. Pearson Education; 2003.
- [3] Eric Newcomer, Understanding Web Services: XML, WSDL, SOAP, and UDDI. Addison-Wesley, (2002)
- [4] World Wide Web Consortium (W3C). <http://www.w3.org/>