Consulting Services

TIBCO Design Patterns

This is a TIBCO Best Practices document from TIBCO Professional Services Group.

This document describes the design patterns generally involved in using TIBCO products and technology. It serves as both a concepts guide and a reference to TIBCO best practice designs.

Document Version: 2005.1

Document Date: 25 Feb 2005

TIBCO® The Power of Now®

http://www.tibco.com

Global Headquarters
3303 Hillview Avenue
Palo Alto, CA 94304
Tel: +1 650-846-1000
Toll Free: 1 800-420-8450
Fax: +1 650-846-1005

For more information, please contact:

TIBCO Software Inc.
3303 Hillview Avenue
Palo Alto, CA 94304
USA

Table of Contents

Table of Figures

Table of Tables

1 Introduction

1.1 Overview

This document presents a set of reusable design and implementation patterns, which can be used to build solutions to solve system integration problems that range from sharing data between disparate systems to managing complex business processes across multiple applications.

Design patterns are building blocks that can be used to form design solutions. Using patterns in projects can provide many benefits: leveraging a proven solution, cutting down the implementation and maintenance cost, and providing a common vocabulary for team communication. This typically results in better-designed, more scalable architecture and shorter project life cycle.

1.2 Version Applicability

Unless otherwise noted in this document, its content is compatible with and updated with the latest versions of all TIBCO products referred to in this document.

1.3 How to Use This Document

The design patterns cover design issues encountered in design, deployment, and operations, and they are presented independently in this document. Readers of this document are encouraged to jump to any specific topic directly to obtain relevant best practices.

To best utilize the content of this document, we recommend hyperlink all the topics in this document in your best practices portal.

1.4 Other Sources of Best Practices

The following are additional sources of best practices related to the topics covered by this document:

- Enterprise Integration Patterns (book) by Gregor Hohpe

2 Understanding TIBCO Design Patterns

2.1 Pattern Definition

Patterns enable us to document a known *recurring problem* and its *solution* in a particular *context*. There are four keywords (in *italic*) in the above definition. When we define a pattern, we must identify the problem, the solution, and the context in which it occurs. A pattern is cataloged only when it is captured in at least three different systems (known as *Rule of Three* in the pattern community).

2.2 TIBCO Design Patterns

Unlike architectural styles, such as *pipe and filter*, which define solutions at a higher level of abstraction, patterns solve problems with a smaller granularity, such as the object-oriented design patterns from the Gang of Four design patterns book.

TIBCO design patterns solve design and implementation problems in integration scenarios, such as message routing, message interaction, data integrity, system manageability, and application scalability. The use of TIBCO products are emphasized in pattern solutions.

2.3 Design Pattern Catalog

This section lists design patterns collected from different TIBCO implementations. They are by no means exhaustive, and are likely to grow with the growth of customer's own integration experiences.

Messaging Oriented Patterns describe the best practices adopted in traditional EAI space that are used to solve integration problems using message-based middleware products such as TIBCO RV and Enterprise Message Service (EMS).

Metadata Management Patterns provide design guidelines in the XML-based metadata (schema) management space. It solves problems in data integration.

Process Implementation Patterns provide implementation techniques and best practices in business process automation and workflow management space.

Performance Enhancement Patterns provide techniques to leverage messaging middleware in performance-oriented applications.

System Management Patterns describe solutions to manage distributed applications.

The following table is a list of design patterns covered in this version of the document:

Index	Pattern	Category	Chapter Reference
1	Publish-subscribe	***Message Oriented Patterns***	3.1
2	Request-reply		3.2
3	Broadcast Request-reply		3.3
4	Chain of Service Providers		3.4
5	Demultiplexer		3.5
6	Bridge		3.6
7	Adapter		3.7
8	Database Adapter		3.8
9	Replicator		3.9
10	Duplicate Detection		3.10
11	Content-based Router		3.11
12	Standard Data Format	***Metadata Management Patterns***	4.1
13	Standard Wire Format		4.2
14	Canonical Data Schema		4.3
15	Transformer		4.4
16	Data Validation		4.5
17	Integration Broker	***Process Implementation Patterns***	5.1
18	Workflow Manager		5.2
19	Process/Task Delegation		5.3
20	Process Synchronization		5.4
21	Parallel Execution		5.5
22	Sequential Execution		5.6
23	Shared Properties		5.7
24	Global Error Handler		5.8
25	Human Approval		5.9
26	Load Balancer	***Performance Enhancement Patterns***	6.1
27	Cache		6.2
27	Message Based Monitoring	***System Management Patterns***	7.1
28	Application Instrumentation		7.2
29	Integrating Enterprise Management Platforms		7.3
30	Self Recovery		7.4
31	Centralized Logging		7.5
32	Log File Scanning		7.6
33	Automatic Throughput Optimization		7.7
34	Heartbeat Generator		7.8

Table 1 Design Pattern Catalog

2.3.1 What Is Not Covered

This document does not cover design patterns that occur in the traditional RPC (Remote Procedure Call) programming model. A typical example of this is J2EE patterns, which can be found from a variety of other resources.

2.4 Design Pattern Template

Design patterns in this document are described using a template. The template consists of portions presenting various attributes for a given pattern. Each pattern starts with its descriptive name and followed by the following sections:

- ***Context***

Set the environment and use cases under which the pattern exists, more precisely, the types of applications that this pattern is applicable.

- ***Problem***

  Describe the design issues faced by the developer, and the problems associated with using existing approaches to solve these issues, and the motivation for better alternatives.

- ***Solution***

  Describe the components that form the solution, implementation details, and design variations. Sample code snippets will be included wherever makes sense. In addition, anti-pattern describes behaviors where the pattern is NOT applicable.

- ***Applicability***

  Provide a quick list of use cases where this pattern is applicable (or where it is not).

- ***Related Patterns***

  Relationship to other design patterns, if any.

3 Message Oriented Patterns

Message oriented patterns are design patterns that use message-oriented middleware (MOM). MOM is traditionally used in the EAI (Enterprise Application Integration) space. With the popularity of the messaging technology, especially those that emerge as industry standards, message oriented patterns have proliferated into spaces that use traditional client-server or component-based technology for application development.

One of the standard based messaging technologies offered by TIBCO is Enterprise Message Service (JMS compliant). In addition to TIBCO EMS, there are alternative messaging technologies such as TIBCO Rendezvous, MQ Series etc. TIBCO Rendezvous is important framework to consider since it is used as underlying mechanism for many TIBCO products such as configuration, deployment, and monitoring.

3.1 Publish-Subscribe

Context

Publish-subscribe interactions are driven by events – a publisher makes information available for general distribution, a subscriber is triggered to receive the information upon the arrival of data. Communication flows in one direction.

Example applications include securities data feed handlers publish the latest stock prices to hundreds of traders on a trading floor simultaneously; materials movement systems distribute data to various materials handlers, controllers and tracking systems on a factory floor; inventory levels flow continuously to accounting, purchasing, marketing and other departments in a retail store; bug tracking database immediately sends bug reports and updates to all personnel interested in a particular project; master database publishes updates to a set of Internet mirrors.

Problem

Traditional RPC (Remote Procedure Call) based technologies (such as COM/DCOM, CORBA, JRMI) are popular in demand-driven client/server applications, but not suitable for event-driven applications, especially when the number of applications interested in receiving the events is potentially large.

The following table lists the scenarios where using MOM technology is better than using traditional RPC based technology in publish-subscribe scenarios.

Scenario	Traditional RPC Technology	MOM Technology
The number of subscribers is big. This is a high fan-out situation.	Delivery of information is inefficient. Messages have to be sent multiple times PTP.	Delivery of information is efficient. Messages have to be sent only once using multicast.
Subscribers can join and leave the subscription list dynamically.	Must change the publisher because the tight addressing scheme (using DNS names, IP addresses).	Publisher is unaffected because publish-subscribe is anonymous via subject names.
Data are changed from the source side infrequently and at irregular interval, while subscribers need to get the updates in real time.	Polling techniques (especially “network polling”) are inefficient and waste of system resources.	Messages are delivered (pushed) to subscribers as events in real time as soon as changes happen. This results in most efficient use of system resources.

Table 2 Compare RPC and MOM Technology

Solution

Publish-subscribe applications are event-driven, as opposed to client-driven or demand-driven.

In publish-subscribe interactions, data producers are decoupled from data consumers – they do not coordinate data transmission with each other, except by using the same address names. Producers and consumers communicate with each other using addresses called *subject names*. This addressing mechanism completely decouples applications from each other and makes both the publishers and subscribers anonymous. It further facilitates publishers and subscribers to be added or removed dynamically in a distributed system – the only requirement is to keep the subject names unchanged.

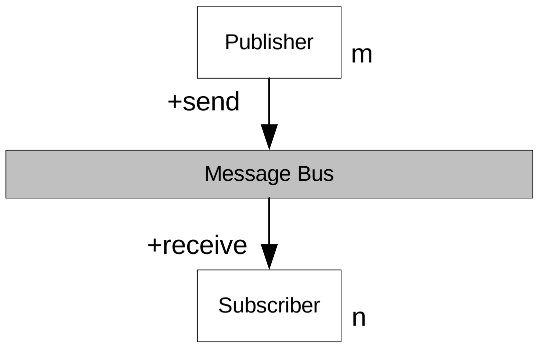


Figure 1 Publish-subscribe Model

The publish-subscribe model consists of m number of publishers and n number of subscribers. Messages are sent from publishers to subscribers via the middleware message bus. The message bus is responsible for registering message distribution list (based on subject name interest) and disseminating messages to their destination. Communication is in one direction (publishers to subscribers), and often one-to-many, but can also be many to many in the most general case.

Implementation

TIBCO BusinessWorks (BW) is the primary integration application development platform that can be used to implement publish-subscribe. The EMS and Rendezvous palettes in BW provide the following configurable publish-subscribe interfaces:

- Publish JMS/Rendezvous Message activity (Publish)
- Wait for JMS/Rendezvous Message activity (Subscribe)
- JMS/Rendezvous Subscriber process starter (Subscribe)

In JMS the messages are transported over Topics whereas in RV through Subjects.

In addition, different TIBCO Adapters such as ADB or Files Adapter can be configured to provide Publication or Subscription Service. The ActiveEnterprise Adapter palette in BW contains activities to communicate with configured TIBCO ActiveEnterprise adapters using publish-subscribe:

- Publish to Adapter activity (Publish)
- Adapter Subscriber process starter (Subscribe)
- Wait for Adapter Message activity (Subscribe)

When applications that use publish-subscribe must be written in programming languages (such as Java or C++), or run within a hosting application (such as application or web server), we cannot use packaged TIBCO products such as BW or shrink-wrapped AE adapters. This is typical in building multi-tiered web applications. Instead, EMS and RV API can be used to build such patterns. Both products include sample publish-subscribe sample programs in its package.

Alternatively, one can use BusinessWorks to subscribe to messages by using the packaged plug-ins. Different AE adapters such as ADB and Files adapters can also be configured to publish or subscribe to JMS/RV.

The following discusses the quality of service (QoS) associated with using EMS or RV messaging to implement publish-subscribe.

RV - Using Different Messaging Quality of Service

Certified message delivery (RVCM) is a QoS (Quality of Service) provided by RV in addition to the default RV reliable message delivery. The following table compares RVCM to RV.

Aspect	**Reliable Delivery**	**Certified Delivery**
Location of Protocols	Reliable message delivery protocols are implemented in the Rendezvous daemon (rvd).	Certified message delivery protocols are implemented in a separate library layer (tibrvcm). This library uses rvd as message transport.
Protocol Visibility	Reliable delivery protocols are invisible to programmers.	Certified delivery calls automatically adhere to certified delivery protocols, yet the protocols give programmers abundant status information and limited control.
Protocol Information	Rendezvous daemons inform programs when data is lost. No information about the lost data is available.	The library presents advisory messages to inform programs of every significant event related to certified delivery. Advisories identify specific messages by correspondent name, subject name and sequence number.
Ledger	None.	The certified delivery library records outbound messages in a ledger, either within the program process storage, or in file storage.
Time Limit	rvd retains outbound messages for 60 seconds.	The certified delivery library retains outbound messages in the ledger until either delivery is complete or the time limit (set by the program) expires.
Effective Range	60 seconds, or rvd process termination—whichever is first.	With persistent correspondents, certified delivery can extend beyond program process restart. It is not affected by rvd process termination.
Network Bandwidth	Minimal network overhead beyond the message itself.	Additional network overhead to confirm delivery of each certified message.
File Storage	No file storage overhead.	Optional file-based ledgers consume file storage for each message until delivery is complete (or the time limit expires).
Routing Daemons	Both protocols work across Rendezvous routing daemons (rvrd).	

Table 3 RV vs. RVCM

Use RVCM in cases where messages must guarantee delivery to subscribers, especially when messages must travel through unreliable networks to reach the subscribers (such as WAN), or subscribers could be down (e.g., for maintenance) when messages are being published continuously.

Using RVCM comes with a price. The messaging throughput of RVCM is typically an order of magnitude lower than that of RV.

JMS - Using Different Messaging Quality of Service

TIBCO EMS provides extended set of Quality of Service for JMS comparable to RV and RVCM. In addition to standard Persistent and Non-Persistent delivery modes for topics in JMS, the TIBCO EMS enables Reliable Message Delivery and No-Acknowledgement Message Receipt modes to achieve better performance of messaging infrastructure. The following table describes all Quality of Service for TIBCO EMS:

Delivery Mode	Description	Standard
Non-Persistent	Non-Persistent messages are never written to persistent storage.	JMS
Persistent	Persistent messages are logged to persistent storage when they are sent. Messages with the persistent delivery mode are always written to persistent storage, except when they are published to a topic that has no durable subscribers. When a topic has no durable subscribers, there are no subscribers that need messages resent in the event of a server failure. Therefore, messages do not need to be saved, and performance is improved because disk I/O is not required	JMS
Durable	There can be a time dependency in the publish and subscribe model. By default, subscribers only receive messages when they are active. If messages are delivered when the subscriber is not available, the subscriber does not receive those messages. JMS specifies a way to remove part of the timing dependency by allowing subscribers to create durable subscriptions. Messages for durable subscriptions are stored on the server until the message expires or the storage limit is reached. **Benefits:** Subscribers can receive messages from a durable subscription even if the subscriber was not available when the message was originally delivered.	JMS
Reliable	PERSISTENT and NON_PERSISTENT delivery require the TIBCO Enterprise Message Service server to return a system message to the client application to ensure proper handling of messages. Using TIBCO RELIABLE_DELIVERY mode precludes the TIBCO Enterprise Message Service server from sending this system message. In reliable delivery mode, the client application does not have to wait for this system message because it will not be sent. **Benefits:** This allows higher message volume. Reliable mode decreases the volume of message traffic, allowing better usage of system resources, and higher message rates.	TIBCO Extension to JMS
No-	In no-acknowledge receipt mode, after the server sends a	TIBCO Extension to

Acknowledgement receipt mode	message to the client, all information regarding that message for that consumer is eliminated from the server. Therefore, there is no need for the client application to send an acknowledgement to the server about the received message. **Benefits:** Not sending acknowledgements decreases the message traffic and saves time for the receiver, therefore allowing better utilization of system resources.	JMS

Table 4 TIBCO EMS QoS

The QoS is set by the message sender or publisher in the *JMSDeliveryMode* message header field.

Interface Definition

The following information must be documented in order to define the interface between the publisher and the subscriber at design time:

Name	**Format or Possible Values**	**Defined by**
RV-Subject or JMS-Topic name	Conform to EAI Subject Naming Standard	Integration Center of Competency (ICC)
Message Schema	XSD (preferred) or AE Schema	ICC or Data Architects
Quality of Service	RV, RVCM, RVDQ, JMS (Non-persistent, Persistent, Durable, Reliable Delivery, No-Acknowledgement receipt)	ICC

Table 5 Publisher Interface Definition

Applicability

Use Publish-subscribe pattern

- When there is more than one subscriber that needs to receive the same information
- When you want to decouple senders and receivers of the same information
- When network multicast is available in your networking infrastructure and you want to take advantage of it to deliver messages more efficiently
- When your application is inherently event-driven, such as applications subscribing to alerts, notifications, and real-time updates

Related Patterns

Request-reply

3.2 Request-Reply

Context

Request-reply applications are client-driven or demand-driven. A client requests data from a server; the server computes an individual response and returns it to the client. Communication flows in both directions.

Demand driven computing is well suited for applications such as transaction processing (as in ATM banking), database query, and web-based query. New standard such as Web Services is also based on demand-driven service oriented architecture (SOA).

Problem

Applications built with traditional RPC (Remote Procedure Call) based technologies have run into performance and scalability issues. In order to scale up the server performance to serve a large number of simultaneous client requests, the server has to be multithreaded at the application level; similarly, client programs cannot send multiple requests in parallel because of the synchronous nature of the RPC protocol, unless it is also multithreaded. Multithreaded applications are considerably more complex, and applications built with traditional RPC based technologies are difficult to maintain.

Solution

To address the challenges of building more efficient distributed applications and service based architecture that are both easy to maintain and scale, the communication model between a client and a server should be asynchronous, which can be built on top of the message oriented middleware (MOM). In fact, MOM has existed even before the distributed component technology was born.

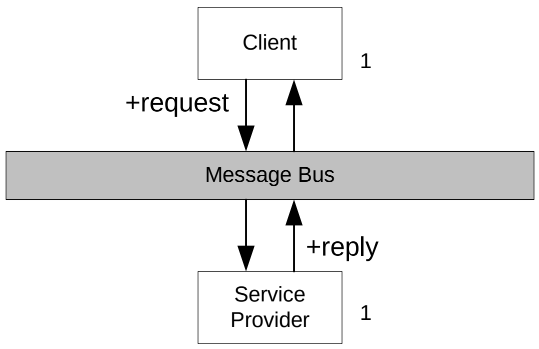


Figure 2 Request-reply Model

MOM based request-reply closely resembles existing RPC based model, but can be either synchronous or asynchronous. The request-reply interaction consists of two messages in a round trip – a request and a reply. Communication is in both directions, and has a one-to-one relationship between the client and the service provider.

In synchronous request-reply scenario, client’s current thread of control blocks on the request until reply comes back or timeout value after an acceptable window of time.

In asynchronous request-reply scenario, client sends request and can continue to execute other tasks without blocking on the request. To get the reply, the client can either wait on the reply explicitly, or exits the current thread of control and uses a different thread of control to asynchronously listen on the reply event or the timeout event when the acceptable window of time expires.

When request-reply is asynchronous, and the reply subject is a broadcast subject name (not a point-to-point or INBOX subject name in RV term), the client must be able to keep track of whether the reply message is meant for itself. This can be implemented by adding a globally unique tracking ID in the request message (one can use a GUID – globally unique identifier – generator to generate a string for this field). When the service provider replies, it includes the tracking ID in the reply message. In JMS case this is achieved through proper update and tracking of JMSMessageID and JMSCorrelationID fields in a JMS message header.

Exception handling is also very important in asynchronous request-reply scenario. The client must be able to handle the timeout exception properly. The service provider must be able to propagate the exception back to the client in messages as well, usually on a different but well-known reply subject name.

Implementation
TIBCO BusinessWorks (BW) is the primary integration application development platform that can be used to implement request-reply. The EMS, Web Services and Rendezvous palettes in BW provide the following configurable request-reply interfaces:

Client**:**

Synchronous request-reply can be implemented using Send Rendezvous Request activity (if use RV) or JMS Queue Requestor (if use JMS). These are the blocking calls.

Asynchronous request-reply can be implemented using Publish Rendezvous Message (JMS Topic Publisher for JMS) activity to send the asynchronous request, and using Wait for Rendezvous Message (Wait for JMS Topic Message for JMS) activity to wait for reply explicitly in the same thread of control (as shown in the following figure), or use the Rendezvous Subscriber (JMS Topic Subscriber for JMS) process starter to wait for reply or timeout event in a separate thread of control. Note that when using the Wait for Rendezvous Message (or Wait for JMS Topic Message for JMS) activity, the reply message can come back before "Execute other tasks" completes. In this case, you can specify the time this reply message will wait before the Wait for Rendezvous Message (or Wait for JMS Topic Message for JMS) activity is executed in the "Event Timeout" configuration field.

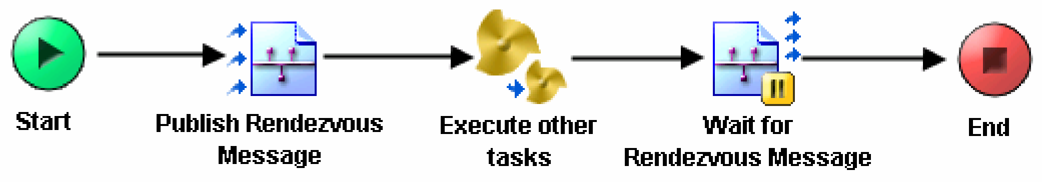


Figure 3 RV - Generic Asynchronous Request-Reply Implementation Using BW

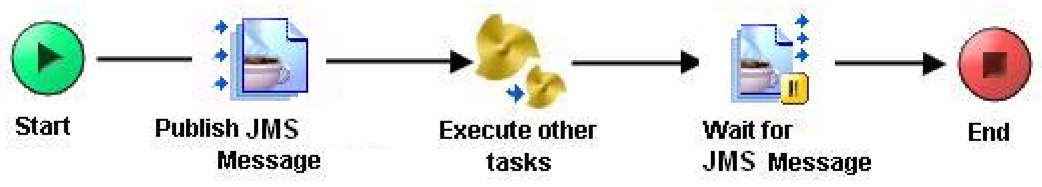


Figure 4 JMS - Generic Asynchronous Request-Reply Implementation Using BW

Service Provider:

Reply to Rendezvous Request activity (service provider to reply to client request – this activity must be preceded by the Wait for Rendezvous Message activity or the Rendezvous Subscriber process starter – see the following figure)

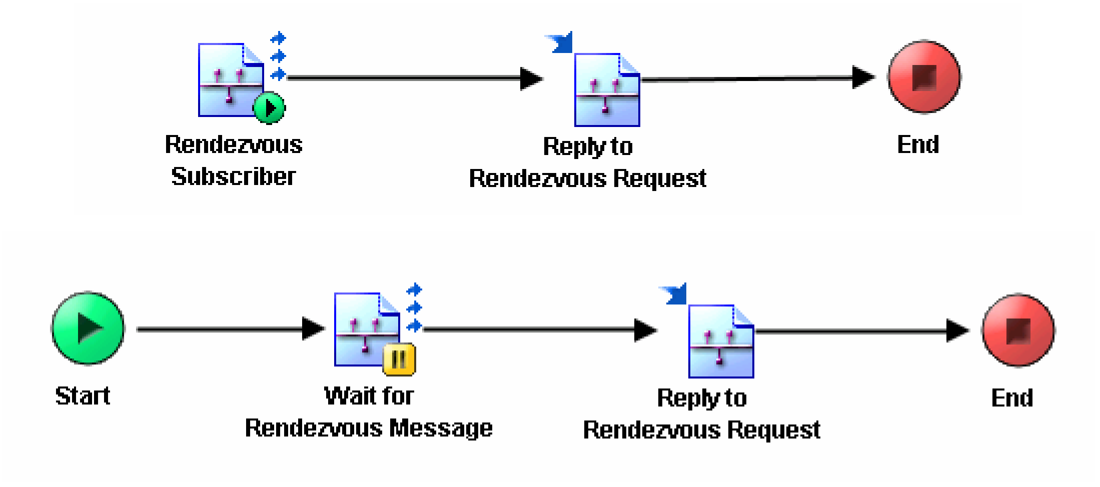

Rendezvous Subscriber
Reply to Rendezvous Request
End
Start
Wait for Rendezvous Message
Reply to Rendezvous Request
End

Figure 5 Generic Service Provider Implementation Using BW

Reply to JMS Request looks similarly.

In addition, different TIBCO ActiveEnterprise (AE) adapters such as ADB or COM adapter can be configured to provide Request-Response Service. The AE Adapter palette in BW contains activities to communicate with configured TIBCO Adapters using request-reply, similar to those available in the BW Rendezvous palette:

<u>Client</u>:

Invoke an Adapter Request-Response activity (implements synchronous request-reply to AE adapters)

The following figure illustrates asynchronous request-reply to AE adapters:

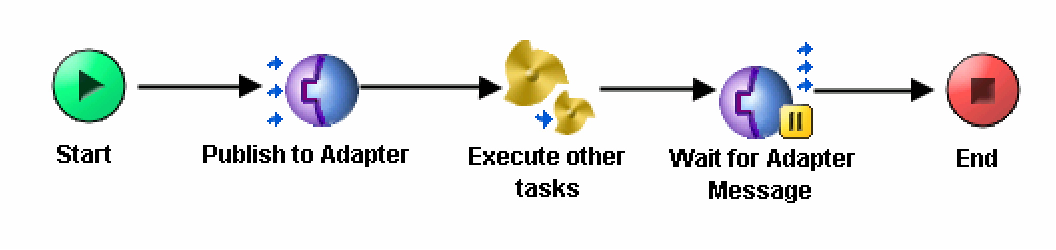
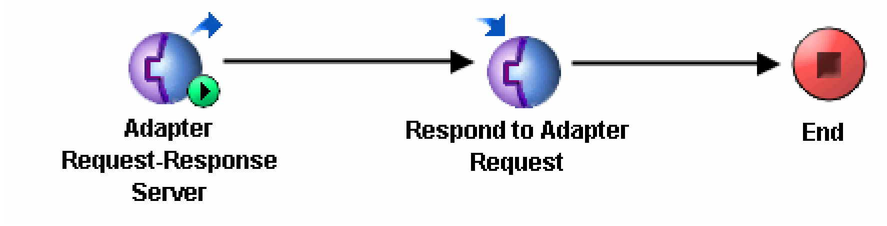


Figure 6 Asynchronous Request-Reply to AE Adapter Request-Response Service

<u>Service Provider:</u>



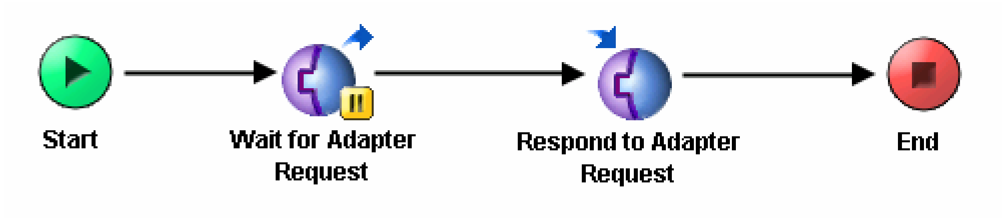


Figure 7 Service Provider to AE Adapter Request-Reply

Another interesting request-reply service provided by BW implements SOAP request-reply (which could be used to invoke a Web Service) using the SOAP Request Reply activity in BW's SOAP palate. Note that the SOAP Request Reply activity is always synchronous.

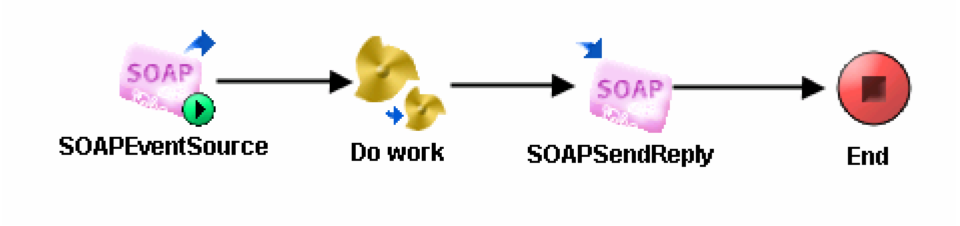


Figure 8 Web Service as Service Provider Using BW

When applications that use request-reply must be written in programming languages (such as Java or C++), or run within a hosting application (such as Web Server or Application Server), we cannot use packaged TIBCO products such as BW or shrink-wrapped AE adapters. This is typical in building multi-tiered web applications. Instead, we recommend using RV or EMS API. Both products come with sample programs implementing request-reply pattern.

When using RV as the transport one thing that developers should realize is that the asynchronous request-reply implemented using BW are not point-to-point messages. They are in essence two publish-subscribe multicast/broadcast RV messages between the client and the service provider. If point-to-point request-reply implementation is desirable (e.g. to limit multicast/broadcast traffic in the network), one must use RV API directly.

The next table shows how to do request-reply using RV API directly:

Step	Client	Service Provider
	Point-to-point	Point-to-point
1	Query server to get server inbox address and use it as the send address setSendSubject()	
2	Set inbox reply subject in the request message setReplySubject()	
3	send() send inbox request message (point-to-point)	
4		Receive client request message in callback
5		sendReply() send response message on client inbox subject (point-to-point)
6	Receive server response message in callback	
	Multicast	Point-to-point
1	Set inbox reply subject in the request message	

	setReplySubject()	
2	send() multicast client request message	
3		Receive client request message in callback
4		sendReply() send response message on client inbox subject (point-to-point)
5	Receive server response message in callback	
	Multicast	Multicast
1	Set regular reply subject in the request message setReplySubject()	
2	send() multicast client request message	
3		Receive client request message in callback
4		sendReply() send response message on client reply subject (multicast)
5	Receive server response message in callback	
	Multicast	Point-to-point
1	sendRequest() send request message and block (on timer if set)	
2		Receive client request message in callback
3		sendReply() send response message on client inbox subject (point-to-point) – note that the client inbox subject is created implicitly by the client-side sendRequest() call
4	Receive server response message as the return value from sendRequest(), no callback	
	Point-to-point	Point-to-point
1	Query server to get server inbox address and use it as the send address setSendSubject()	
2	sendRequest() send request message and block (on timer if set)	
3		Receive client request message in callback
4		sendReply() send response message on client inbox subject (point-to-point) – note that the client inbox subject is created implicitly by the client-side sendRequest() call
5	Receive server response message as the return value from sendRequest(), no callback	

Table 6 Request-reply Using RV API

The next table shows how to do request-reply using EMS API directly:

Step	Client	Service Provider
1	Create connection factory: com.tibco.tibjms.TibjmsQueueConnectionFactory(serverUrl)	Similar
2	Create connection:	similar

	factory.createQueueConnection(userName,password)	
3	Create connection session: connection.createQueueSession()	similar
4	Create destination: session.createQueue()	similar
5	Create sender: session.createSender()	Create both receiver and sender (for reply): session.createSender() session.createReceiver()
6	Send message: Sender.send(destination, msg)	-
7		Receive client request message: Receiver.receive()
8		Send response message: Sender.send(destination, reply_msg)

Table 7 Request-reply Using TIBCO JMS API (using queue)

Interface Definition

In order to use the request-reply pattern, one must define the interface between the client and the service provider at design time. The following information must be provided for each client interface:

Name	**Format or Possible Values**	**Responsibility**
Subject name	Conform to EAI Subject Naming Standard	ICC
Message Schema	XSD (preferred) or AE Schema	ICC or Data Architects
Quality of Service	RV, RVCM, RVDQ, JMS (Non-persistent, Persistent, Durable, Reliable Delivery, No-Acknowledgement receipt)	ICC

Table 8 Client Interface Definition

Similarly, the same information should be provided for each service provider interface. In asynchronous request-reply, you always have to specify a reply interface.

Applicability

Use Request-reply pattern when:

- You want to improve your server performance and make it more scalable
- You want to improve your client performance by using asynchronous calls to the server and execute tasks in parallel
- You need explicit acknowledgement from the service provider (at the application level, vs. at the message protocol level)

Related Patterns

Publish-subscribe

3.3 Broadcast Request-Reply

Context

More than one service providers can handle the request. The client can request services provided by multiple service providers without involving additional cost. While traditional request-reply interactions involve one client and one server, in broadcast request-reply interactions multiple servers can receive the request and reply as appropriate.

This pattern is useful in cases where multiple service providers compete to provide the same service, or when the client wants to receive information from multiple service providers, such as network management applications that broadcast requests for test information. It is also useful when there is not an effective way to utilize the redundancy provided by multiple servers, for example, database query with load balancing among multiple servers, and distribution of computing sub-tasks to the first available server.

Problem

When multiple service providers are available to provide service to one client request, the client needs to contact each service provider one at a time to fully utilize the services provided by all. There are other times where servers are added and deleted dynamically. To reflect the changes, client programs are usually required to change when using the traditional RPC (Remote Procedure Call) based programming model.

Solution

Broadcast request-reply provides a simple yet effective way of accessing the service provided by multiple servers in a single request.

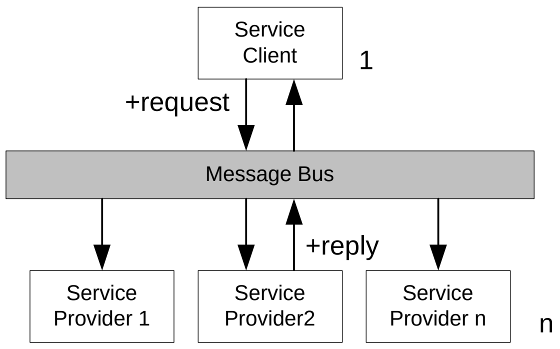


Figure 9 Broadcast Request-reply Model

In this pattern, the client publishes the request in a broadcast message. Communication flows in both directions, and only some service providers respond to the client request. Depending on the types of services service providers provide and how the client decides to utilize the replies, the client can process one or multiple replies based on pre-defined policies. One commonly used policy is that the client takes only the first good reply, and ignores the rest. It is also possible that the client takes more than one reply and compares the result for accuracy. Yet another policy can be taking many replies and aggregate the result. The complete interaction consists of one broadcast or multicast request message, and any number of reply messages.

Implementation

All service providers participating in this pattern must register the same subject name for the same service. The request is a multicast or broadcast message, and the requests from different service providers should be implemented as point-to-point reply messages. The BW implementation of this pattern is similar to the publish-subscribe implementation. The Broadcast Request-reply pattern can be implemented in EMS or RV API.

Applicability
Consider using Broadcast Request-reply pattern when multiple servers exist to serve the same client request.

Related Patterns
Publish-subscribe, Request-reply, Demultiplexer, Load Balancer

3.4 Chain of Service Providers

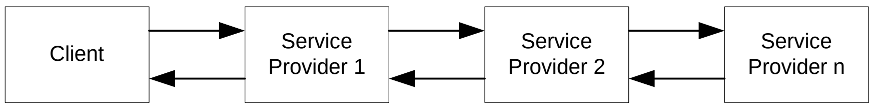
Context

A service request must be processed by multiple service providers that delegate among each other in the chain.

Problem

A service provider isn't always able to handle a request by itself. For example, service provider A may need to forward the request to the next service provider B for processing or additional service before A can reply. Traditional methods implementing this pattern using RPC (Remote Procedure Call) model pass requests through the chain with blocking calls. This performance degrades severely when the number of service providers in the chain grows.

Solution



(Note: message bus is not shown)

Figure 10 Chain of Service Providers

The Chain of Service Providers (CSP in short) system pattern is used to *delegate* or *redirect* service in an asynchronous way. It is used to build delegation servers to process service requests that cannot be processed by a single server. The client is not aware of the additional services provided by the chain of service providers.

Implementation

In essence, CSP consists of a series of single request-reply interactions to serve the original client request, but the interaction between each request and reply is point-to-point in nature. Consequently, a queue-based messaging solution is more suitable to implement the CSP pattern. RV uses a publish-subscribe based messaging protocol. Although RV can be used to implement this pattern (using its point-to-point API), a better alternative is to use a chain of request-replies pattern based on JMS queues.

In addition, in order to track messages passed asynchronously through the chain of service providers, we must be able to track individual reply intended for the original request. We include a stack variable in each request message that contains a stack of reply subject name and message tracking ID pairs. The following figure illustrates this idea:

reply subject for service provider n-1	msg tracking ID n-1
...	...
reply subject for service provider 2	msg tracking ID 3
reply subject for service provider 1	msg tracking ID 2
reply subject for client	msg tracking ID 1

Figure 11 Stack Variable Used in CSP Pattern

Applicability

Use Chain of Service Providers pattern when a service is provided by more than one server in a chain.

Related Patterns

Request-reply

3.5 Demultiplexer

Context

A client wants to request from multiple service providers or invokes multiple requests from the same server within the context of one use case, and doesn't require an immediate response from each request. The client only wants the final result back based on the replies from all requests.

Consider a portal application. The HTML content of the portal web page is returned from multiple content providers from the backend.

Consider a web-based order management system. An order entry must be serviced by multiple backend service providers such as credit check, inventory, and tax and shipping charge calculation in certain sequence. The result of the multiple services is to be returned as if they were executed in a single transaction.

Consider a simple web-based airline registration system where an Application Server receives a request to reserve a seat for a user for a particular flight. In this scenario, the Application Server must register a user with an airline, determine if seats are available on a flight and if so, reserve a seat for a user, as illustrated in the following figure (the example was taken from an EJB implementation, hence the EJB terms such as *Home* and *PrimaryKey* – but the concept applies to any application server):

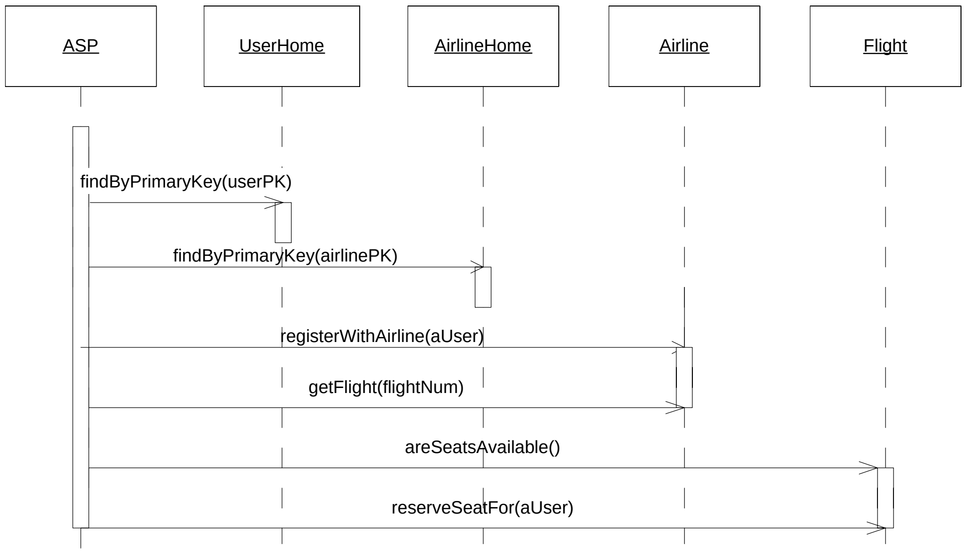


Figure 12 Airline Reservation Example

Problem

The execution of above use cases suffers several drawbacks:

- Poor performance and slow response time: multiple synchronous RPC calls block on each call, in addition to the overhead of security (e.g. to authenticate and encrypt messages when crossing firewalls) and transaction management (on the part of the service provider's execution environment)
- Lack of transaction consistency: since multiple client requests could result into multiple transactions executed separately by any number of the service providers, the state of the system could be left into inconsistency. It

would require the client to implement compensating transactions based on exceptions propagated from each reply. Such implementations could be very expensive

- Tight coupling: the client is coupled with many service providers. A change in any service provider could result in change of the client
- Thick/fat client implementation: the business logic that involves the sequence of invoking multiple service providers are trapped inside of the client. Reusability and client code maintainability are low

Solution

The client will send an asynchronous, compound request message to a demultiplexer. The demultiplexer system pattern is used to split the compound request into multiple requests and further invoke individual requests locally in a single transaction (if all requests can be served by the same service provider), or send the requests to appropriate service providers. The demultiplexer is also responsible to construct the results from all replies and send them back to the client in a single reply message. The following figure illustrates this idea:

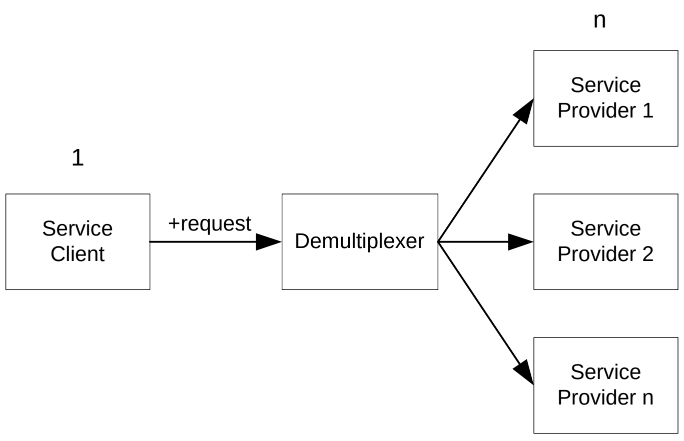


Figure 13 Demultiplexer

The demultiplexer solves all of the problems presented earlier in this discussion.

- Better performance and faster response time because there are less messages travels between the client and the service providers.
- Transactional messaging can be used by the demultiplexer to roll back a client request if any of the requests demultiplexed to different service providers fails, and the client request message can be retried later without the knowledge of the client. Thus the complex transaction management is hidden from the client and managed by the multiplexer. Note that in order to use transactional messaging, we need to use the RVTX (RV Transactional Messaging) product or EMS (JMS).
- The client is totally decoupled from the service providers. The demultiplexer provides the layer of abstraction using subject name based messaging interface.
- Business logic of invoking multiple service providers, its sequence (serial or parallel depending on business logic), and the interface of each service provider, are all moved into the demultiplexer. Client implementation becomes much thinner.

The following figure illustrates the airline reservation use case using the demultiplexer (the example was taken from an EJB implementation, hence the EJB terms such as *MDB*, *Home*, *PrimaryKey* – but the concept applies to any application server):

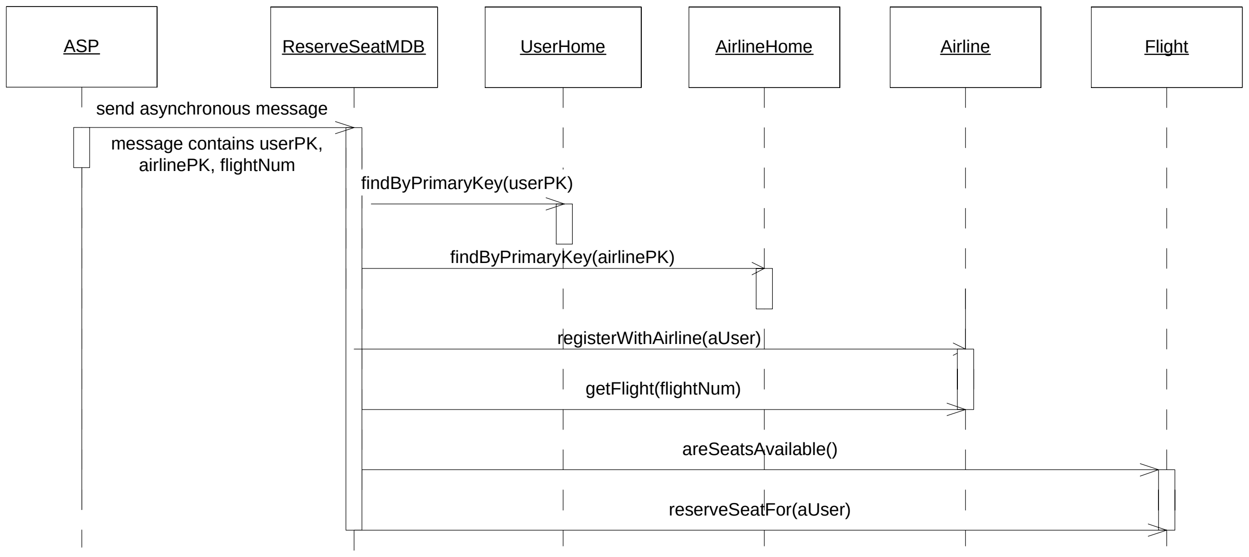


Figure 14 Airline Reservation Example Using Demultiplexer

Implementation

BW is the preferred platform to implement the demultiplexer pattern when the service requests from the client are provided by different backend service providers (such as different legacy applications running independently).

However, if the client requests can be served by components running in the same execution environment (for example, an EJB application running on a J2EE application server), the demultiplexer pattern can be implemented using the asynchronous request-reply messaging API and deployed as part of the service provider execution environment. This allows the demultiplexer to execute more efficiently and to take advantage of the component services provided by the execution environment.

Applicability

Use the Demultiplexer pattern when you want to build a scalable, decoupled server that can process compound client request.

Related Patterns

Request-reply

3.6 Bridge

Context

There exist many different messaging systems (such as TCP, HTTP, and various message-oriented middleware) to transfer data on the network. Messaging is made available to applications via message transport. A message transport is the communication endpoints of the applications. In addition, each messaging system specifies the format of data carried in the message payload.

For example, RV uses a UDP based protocol invented by TIBCO called TRDP (TIBCO Reliable Datagram Protocol), and the message format used by RV to represent its message payload is called RvMsg (RvMsg supports both simple and complex binary data types).

Different applications use different messaging transports to send data for inter-process communication. For example, Java applications use RMI, TIBCO AE (ActiveEnterprise) Adapters use RV/JMS, and applications supporting web services use SOAP to send messages between clients and service providers.

Problem

A distributed enterprise environment could deploy many different applications that use different message transports. This makes exchanging information between applications using different transports difficult. For example, a COM application cannot transfer data easily to an application that only uses SOAP transport to receive data.

Solution

Wire format is the data structure used by the messaging middleware to represent application data payload (see EIF Metadata Management Patterns -- Standard Wire Format system pattern for a more complete definition on wire format). Bridge is a design pattern that is used to convert between two different wire formats.

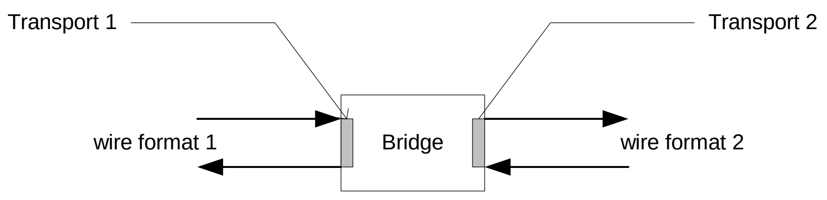


Figure 15 Bridge Model

Implementation

BW is a mega-bridge that provides transports to many different messaging systems. For example, BW can be used to translate a SOAP request into a RV or JMS request message. BW can also serve as a HTTP server that translates a HTTP request into a JMS/RV request message.

Alternatively, bridges can be implemented using custom code if performance is the highest priority. However, BW is very easy to use to implement bridges because it supports many different transports natively.

We recommend using BW as the generic platform to implement the bridge pattern.

RV to JMS Bridge

Since RV is a significant part of TIBCO product family deployment it makes sense to consider closely the bridging mechanisms between RV and EMS (JMS). TIBCO's EMS (JMS) server provides built-in bridging functionalities between RV and JMS. However, since JMS messages from different JMS providers (such as those from IBM or BEA) are not compatible on the wire, it is more flexible to use BW to implement the bridging functionalities, as the built-in RV-JMS bridge in EMS only bridges TIBCO's JMS to RV. The following figure illustrates this implementation.

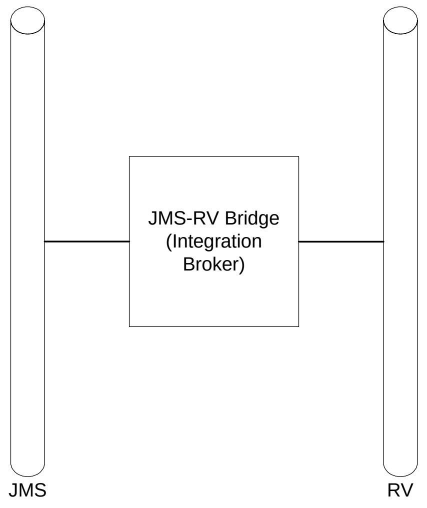


Figure 16 RV-JMS bridge

To build bridge between RV and JMS, the following messaging protocol semantics must be observed when bridging RV and JMS messages:

- Import RV or RVCM messages to JMS topic
- Export JMS messages to RV or RVCM messages
- Import RV or RVCM messages to JMS queue
- Export JMS queue messages to RV messages is not allowed (no semantic mapping between the two). As a result, request-reply messages sent from RV to JMS queue can be tricky, as reply is not allowed from a JMS queue to RV

In addition, we need to pay attention to the following message format differences:

EMS can translate any message type into TIBCO Rendezvous message, but the MapMessage type works best for this translation:

- JMSBytes -> TIBRVMSG_OPAQUE
- JMSObject -> TIBRVMSG_OPAQUE
- JMSStream -> TIBRVMSG_OPAQUE
- JMSText -> TIBRVMSG_STRING
- JMSMapMessage: its fields are mapped directly into top-level TIBCO Rendezvous message fields

EMS extends JMSMapMessage to have array and sub-message types, similar to those found in RV. JMS header and property are translated into sub-messages in RV message, with the following restriction:

- JMS property name length is not limited, while the names of TIBCO Rendezvous fields are limited to 127 characters. If your JMS message has a property name longer than 127 characters, that property field will not be exported

Headers and properties can be optionally exported to RV by setting properties in configuration file.

Applicability

Use Bridge pattern when you need to convert messages from other messaging system to RV/JMS.

Related Patterns

Adapter

3.7 Adapter

Context

ERP systems such as Oracle Apps, SAP, or Peoplesoft, CRM systems such as Siebel, SCM systems such as I2, or custom built applications, need to be integrated to exchange information with each other in a best-of-breed enterprise environment.

Problem

The basic task to integrate with a legacy application such as any of the above-mentioned applications is to move data in and out of them without violating transaction integrity. Sometimes, data must be moved in and out of a legacy application using native API; sometimes, it is possible to interact with the legacy application by working directly on its data store (files or databases). No matter which way data is accessed with, each legacy application has its distinct internal data format (binary data structure) that is different than the messaging middleware message format. For example, SAP uses a format called IDoc; others store data in proprietary file format or in relational databases.

Solution

Adapter has different meanings in different context, such as the “adapter” pattern used in object-oriented design patterns. In the context of this discussion, adapter is used to integrate legacy applications to a standard message oriented middleware chosen in an enterprise. This involves moving data in and out of the legacy application from and to the messaging middleware, using legacy application API if available (or required). The following picture illustrates the adapter pattern:

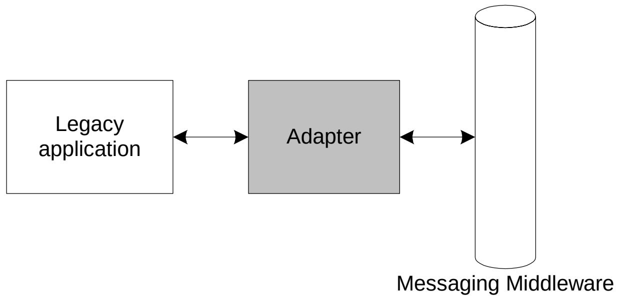


Figure 17 The Adapter Pattern

One of the most important functionality of the adapter is to translate between native application data formats used by legacy applications and the standard messaging middleware message format (RV or JMS wire format). As shown in the following figure, on one end, adapter talks to applications using its native interface (including file and database if no API exists); on the other end, adapter talks to the messaging middleware. Adapter translates the legacy application native data format to and from the middleware application-independent message format.

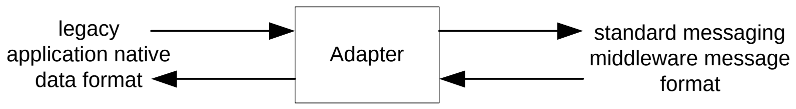

legacy application native data format
Adapter
standard messaging middleware message format

Figure 18 Adapter Functionality

Implementation

TIBCO provides a long list of pre-packaged adapters which include ERP applications, CRM applications, mainframe applications, files, and etc. When no pre-packaged adapters exist, TIBCO provides Adapter SDK (in C++ or Java) for fast and convenient custom adapter development to tap your custom application into the EMS or RV message bus. As an alternative, adapters can be developed directly with JMS or RV API.

Applicability

Use Adapter pattern when you need to integrate a legacy application with JMS or RV.

Related Patterns

Bridge, Database Adapter, Standard Data Format

3.8 Database Adapter

Context

Most enterprise information is stored in databases. Information needs to be retrieved from databases into applications.

Problem

It is desirable to make critical business data changes in databases available to applications as soon as possible. Sound business decisions are made based on the most up-to-date information. Traditional ways of obtaining this information is to query the databases periodically for the most up-to-date information. Database query is usually performed through ODBC or JDBC interfaces.

If multiple applications need to access the same data from the same database, they have to do the same SQL query multiple times. This not only duplicates the DB query effort, but also slows down the entire DBMS system.

In addition, data accessed from databases using ODBC or JDBC cannot be delivered reliably to remote applications.

Solution

Database adapter is specialized adapters that performs a number of database related tasks, such as database query and database transactions, or publish/notify data changes from databases as they occur. It is used so often that it deserves its own pattern from the generic Adapter pattern.

On one end, database adapter interfaces directly with the database using standard DB access methods such as ODBC or JDBC. On the other end, it makes the database information available to the standard messaging bus.

Implementation

TIBCO ActiveDatabase Adapter (ADB) provides the following main features:

1. Database adapters publish data changes (insert/update/delete) from databases on to the middleware bus in near real time, making it available to any application that subscribes to the subject on which the information is published.
2. Use its alerter to send database changes as alerts to monitoring applications, portals, etc.
3. Using request-reply semantics, client applications can send SQL statements, stored procedures, or both on a specified subject to an ADB agent. The agent will process the request and return the results in a reply to the client.

Database adapter is suitable in case 1 and 2. In case 3, it is not necessarily the most suitable database query mechanism. The following table summarizes the differences between using database adapters to query database and using more conventional means such as ODBC/JDBC to query databases.

Compared Features	Direct Query (JDBC, ODBC)	Database Adapter Request-reply Query
Coding/development/ deployment effort	Custom development (Java or C++); easier for small scale quick deployment	Off-the-shelf product, but need configuration of the adapter and infrastructure support for deployment
Performance/scalability	Can achieve high scalability through complex custom coding	Adapter is already optimized for high throughput
Reliability	ODBC or JDBC is not reliable if the query spans across	Query can be performed reliably across unreliable network because of

	unreliable network. ODBC/JDBC is not designed to recover from failure. No effect if query is performed on the same LAN or host	the reliable messaging protocol on which the query is based
Cost on system resources	Dedicated link between the querying app and database throughout the session; could be expensive if query spans across expensive or low-bandwidth network	Asynchronous communication between the DB adapter and the query application on a non-dedicated link. Lower cost if network cost is high
Maintenance/upgrade/ support	Need to be custom maintained	Supported by vendor; upgrade is easier

Table 9 Comparison of Database Query Using ADB and ODBC/JDBC

Design Variation

One of the problems of using ADB to do database query or DB transactions is that the database logic (in the form of SQL statements or store procedure calls) resides on the application side (the client which sends a request to ADB). This makes the application tightly coupled with the database. Should changes occur in the database, the applications could have to update too.

There are two ways to decouple the database logic from the calling applications (whether from within BW or custom applications). The first way is to develop a custom database adapter (say, using ADO or JDBC) to store the database logic in the adapter. The custom database adapter maps a subject name to a piece of predefined database logic, which can be either a block of SQL statements or store procedure calls. As a result, the application only needs to know the message interface to talk to the database. The down side is that custom coding must be done, in many ways recreating the wheel of what ADB has already done.

The alternative is to wrap the database logic in a BW sub-process. Although this stills requires changes at the application end (in the BW sub-process), it decouples the main application logic from the database logic which now resides in a separate BW process.

Despite certain limitations of ADB, it is a robust and appealing tool for DB access because:

- It traps database errors on behalf of the applications
- It hides all the vendor specific DB access stuff from the applications
- It provides scalability out-of-the-box (message-based load balancing and multi-threaded request-reply service)

Applicability

Use Database Adapter pattern when:

- Your application (such as web/portal app) wants to be notified at database changes (update, insertion, delete) in real time
- You must synchronize data from a database in real time
- You guarantee data delivery from a database across the network (usually across an unreliable WAN link) – even in case of network failure
- You need to replicate data from/to databases (see the Replicator pattern)
- You want to separate database logic (e.g. SQL statements) from your applications

Related Patterns

Adapter, Replicator

3.9 Replicator

Context

In a distributed enterprise environment, information resides at many different locations. Files and databases are replicated from one location to the other frequently. There are two most common replications that happen every day. They are database replications and file replications. For example, the UNIX ‘cp’ command is an over-simplified version of a file replicator. FTP is a tool that is used frequently to replicate files all over the places. On the database side, there exist many tools from database vendors to replicate database, such as Oracle Replications and Oracle AQ (Advanced Queue) in Oracle.

Problem

Most of the above-mentioned existing replication methods have two drawbacks: the replication is unreliable, and they are batch-oriented (thus not real-time replication). As a result, information might not be replicated to the desired destination when they are needed.

Solution

Messaging middleware can be used to replicate information from one location to the other. Database adapters are used quite often to replicate database tables across the WAN reliably (see the following figure) using RVCM (Rendezvous Certified Messaging) or JMS. TIBCO ActiveDatabase Adapter (ADB) can replicate DB tables that have multiple levels of parent-child relationships in one RV message. A single JMS or RV message can support as much as 64MB of data payload. This is illustrated in the following diagram.

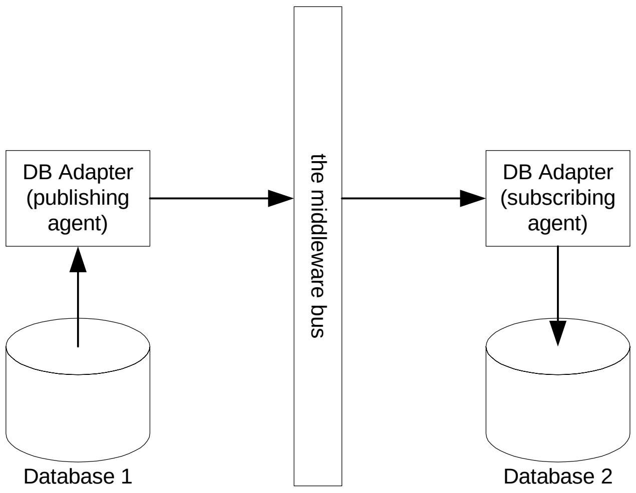


Figure 19 Database Replicator

Another use of the replicator pattern is to actively move files from one location to the other across network boundaries, as an alternative to use FTP, NFS mounts, mapped drives, or other system utilities. The next figure illustrates an implementation of the file replicator.

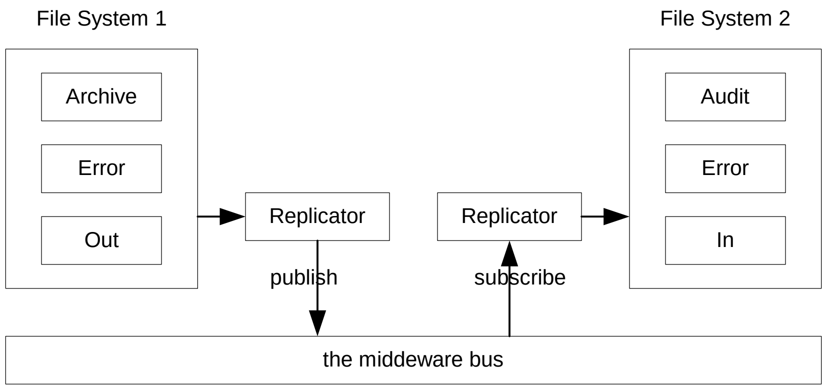


Figure 20 File Replicator

In the above figure, when a new file is dropped into the Out directory of file system 1, it is immediately moved to the In directory of file system 2 via chunks of JMS or RV messages using the publish-subscribe pattern. The size of each chunk is configurable and defaulted to the most efficient size for network transmission (usually this number is 1200 bytes). A copy of the file is saved in the Archive directory; the timestamp etc. information is saved in the Audit directory of file system 2. Both file systems have an Error directory to log publication and subscription errors. For example, file replicators can be used to move files from internal systems to the B2B server in the DMZ to replace the batched FTP process.

The benefit of using file replicator over FTP is that it can take advantage of the message bus, hiding any system and network changes, and it is near real-time (depending on the file system polling interval).

TIBCO File Adapter and BW can be used to implement the File Replicator.

Applicability

Use Replicator pattern

- When you want to copy files from one directory to the other as soon as there is a change in the source directory (new file, updated file, and etc.). In addition, the file transfer is guaranteed even in case of network failure.
- When you want to copy data from one database to the other as soon as there is a change in the source database. In addition, the data transfer is guaranteed even in case of network failure.
- When you want to replicate data from ODS (operational data store) database to corporate reporting database or decision support database.

Related Patterns

Database adapter, Publish-subscribe

3.10 Duplicate Detection

Context

RV and JMS have multiple QoS (Quality of Service), including reliable and certified deliveries. JMS and RVCM are commonly used to guarantee message delivery between two applications (typically a publisher and a subscriber).

Problem

RVCM protocol can generate duplicate messages in certain scenarios. For example, when a RVCM acknowledgement message fails to reach the certified publisher from the certified subscriber, the certified publisher thinks that message is lost and would try to resend the message when the subscriber process is restarted, say, after scheduled maintenance. This could happen if the subscriber confirms the certified message explicitly – if the subscriber raises an exception and exits before it completes processing the received message, the certified message could not be confirmed. This could also happen in case of network failure. In either case, the same message could be delivered to the subscriber more than once.

If the subscriber application cannot allow processing the same message multiple times, a duplicate detection mechanism must be implemented.

Even if the subscriber application does allow processing duplicate messages, re-processing the message would be a waste of resources.

Solution

Like its predecessor *IntegrationManager (IM),* BW (5.2) also provides built-in mechanism to detect duplicate messages. The following describes a generic duplicate detection design which is similar to those used in IM and BW.

To make duplicate detection possible, each message must have a key field (message ID) that can be used to uniquely identify the message. One can use a GUID (globally unique identifier) generator to generate a unique string in this field. (GUID generator can be obtained freely from the Internet.) An alternative is to use an existing field that is known to be unique within the application scope, e.g., an OrderID, or a CustomerID. Sometimes, a compound key would do the trick by combining several fields in the message.

However, it is not enough to use just the unique field to identify the message. As messages travel through multiple components (such as in the example of Chain of Service Providers pattern), we need to differentiate the same messages received by different jobs from duplicate messages received by the same job. We can prefix the message ID with the job ID or a timestamp.

The keys must be persisted to check against its uniqueness. The key values can be stored in a <key value>.key file where the filename is based on the key value. However, a more efficient way is to store the keys in a database table and build a duplicate key detection database adapter to serve queries to the key database.

The keys should be allowed to expire after a certain time period (to allow the key database purge those expired keys from time to time).

A duplicate detection BW process is created to handle the duplicate messages. When duplicate keys are detected in a message and the message is confirmable (JMS, RVCM, RVCMQ, or any other events), the message is confirmed automatically. In addition, the detected duplicate event should be logged.

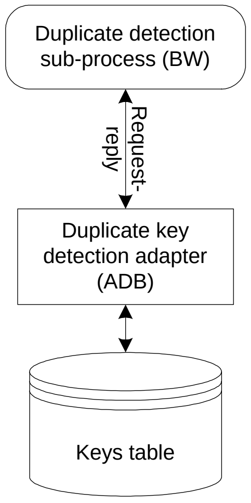

Duplicate detection sub-process (BW)
Request-reply
Duplicate key detection adapter (ADB)
Keys table

Figure 21 Duplicate Detection Component Design

The above figure illustrates the duplicate detection sub-process invoking the duplicate detection key adapter.

In cases where message throughput requirement is very high, say, >100 msgs/second, duplicate detection will adversely impact performance. As such, this implementation should only be used in a highly scalable deployment scheme when duplicate detection is mandated by business requirement.

Applicability
Use Duplicate Detection pattern in all JMS and RVCM subscribers if the subscribers cannot allow processing duplicate messages.

Related Patterns
Publish-subscribe, Standard Data Format, Database Adapter

3.11 Content-Based Router

Context

Messages generated from a single source are processed by different applications based on version numbers, trading partner names, contact numbers, or etc. For example, a “middleman” (usually a B2B service provider or an e-marketplace) trades with ACME Corporation on behalf of multiple trading partners of ACME. ACME wants to re-route these messages based on the names of the trading partners to different backend system for processing because each trading partner has a different contract with ACME.

Problem

The messages generated by the source system must be sent to different destinations based on the content (field values) in the message. However, the source system is not capable of communicating directly with multiple destination interfaces, or simply do not care or is not aware of the multiple destinations the messages must reach.

Solution

Content-based router is used as the filter to re-route source messages to their appropriate destinations.

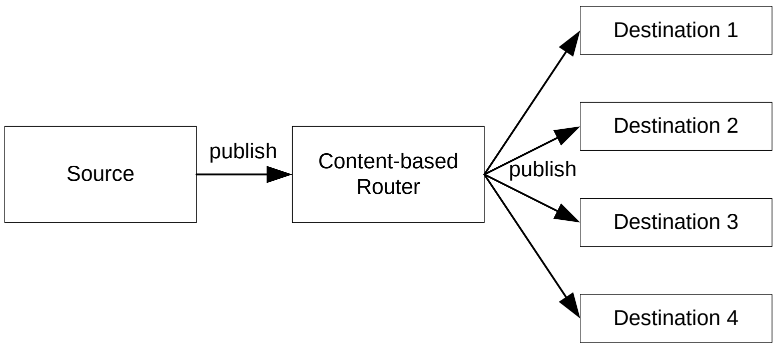


Figure 22 Content-based Router

Content-based router subscribes to messages from the source system, extracts the desired message content, applies a set of pre-defined rules to filter the message into different categories, and re-publish the same messages on different subject names. Content-based router generates the destination address dynamically based on the filtered message content.

Content-based router applies only in publish-subscribe scenario. For request-reply, refer to the demultiplexer system pattern. In addition, content-based router does not modify the message content. For changing message content before republishing, use the Transformer system pattern.

Implementation

BW can be used to implement content-based router. BW provides convenient functionalities with drag-and-drop ease-of-use to filter message content and re-route messages to different destinations. If messages are in standard XML format, extracting message content using Parse XML activity is a very easy task. Destination addresses can be generated dynamically using XPath expressions and feed to publishers.

Applicability

Use Content-based router pattern when messages must be distributed to different destinations based on message content.

Related Patterns

Publish-subscribe, Transformer, Standard Message Format

4 Metadata Management Patterns

Sharing data among disparate applications across the enterprise efficiently is one of the most critical goals in Enterprise Application Integration (EAI).

Metadata management patterns provide data representation and data manipulation techniques to facilitate data sharing between applications.

Although metadata management patterns discussed in this chapter are used most often in Internal EAI applications, they help to define the overall enterprise data architecture and strategies by setting standards and providing techniques to implement these standards.

4.1 Standard Data Format

Context

In a distributed enterprise computing environment that consists of heterogeneous systems, data are represented and stored in a variety of different formats. ***Data format* is the syntactic representation of data.** Many data formats are native to the applications to which they belong. For example, a VB application represents its data in types supported by the Visual Basic language; Excel spreadsheets can be stored in CSV file format (comma separated value – an ASCII file); MS SQL Server manages data in relational data format; TIBCO Rendezvous (RV) software transfers data supported by the RV self-describing data type system (refer to *TIBCO Rendezvous Concepts, Software Release 7.x, Chapter 6 Data* for details); and etc.

Problem

We still live in a world in which legacy applications that represent or store data in many different formats exist. Consequently, one of the biggest problems to solve is how to efficiently convert these different, native data formats among each other when different applications are integrated.

The process of converting data from one data format to the other is called *data translation*.

When more than three applications need to be integrated and each application has a different data format, data translation between each pair escalates into an O(n^2) problem. The total number of translations that one could build for n systems with distinct data formats is n*(n-1).

Solution

The solution is simple: pick or define a standard data format and build (new) applications that support and use it natively. For existing (legacy) applications, build data translations to convert the application native data format into the standard data format. By choosing a standard data format, the total number of data translations can be reduced to O(n).

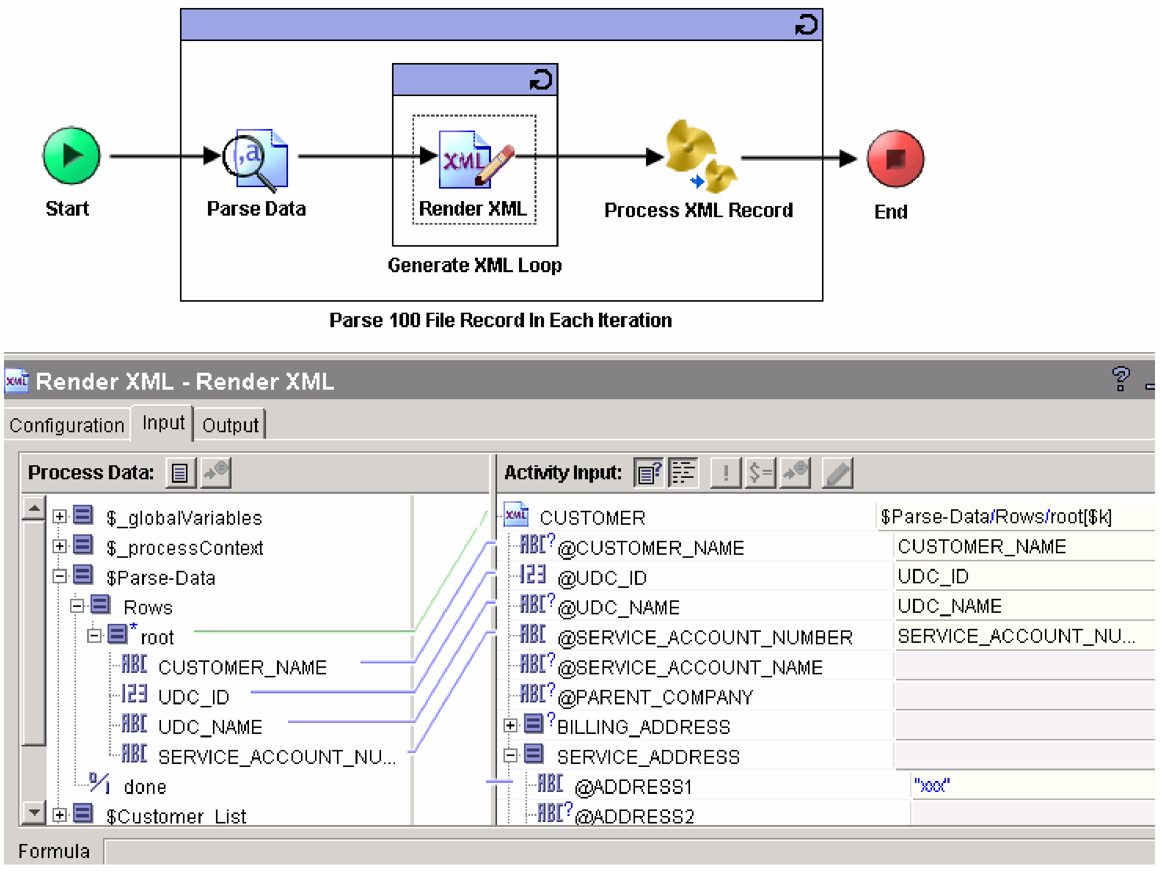
Although selecting the messaging middleware wire format (see Standard Wire Format system pattern in section 3.2) as the standard data format for all applications will make data transfer the most efficient, as there is no need to translate data between the application data format and its wire format, an important design goal is to separate the wire format, thus the messaging transport (such as RV, JMS, SOAP, and etc.), from the standard data format used by applications. This will allow the use of different messaging transport in the future without affecting the applications.

XML is a data format that has become the de facto standard for representing information and business objects in enterprise computing. Because XML is a structured data format, the content and structure of documents can be validated to be sure that all of the required information is provided in the correct format. XML is one of the most important technologies for business integration both inside and between enterprises because it makes it much easier to share data and business objects between incompatible systems.

To facilitate new application development, TIBCO has incorporated comprehensive XML management functions into various products. For example, TIBCO Extensibility products support the entire life cycle management of XML documents. BusinessWorks (BW) supports XML rendering, parsing and validation.

Implementation

A common task in integration is to translate Fixed Record file content into XML documents. One thing that must be kept in mind is that the size of the XML document (in String format) is big, compared with its original form in fixed record file format. As a result, when processing large files with thousands of records or more, it is recommended to process a limited number of records at a time. The following implementation illustrates this implementation in BW:



Process Data:

- $_globalVariables
- $_processContext
- $Parse-Data
  - Rows
    - root
      - CUSTOMER_NAME
      - UDC_ID
      - UDC_NAME
      - SERVICE_ACCOUNT_NU...
    - done
- $Customer_List

Activity Input:

CUSTOMER	$Parse-Data/Rows/root[$k]
@CUSTOMER_NAME	CUSTOMER_NAME
@UDC_ID	UDC_ID
@UDC_NAME	UDC_NAME
@SERVICE_ACCOUNT_NUMBER	SERVICE_ACCOUNT_NU...
@SERVICE_ACCOUNT_NAME	
@PARENT_COMPANY	
BILLING_ADDRESS	
SERVICE_ADDRESS	
@ADDRESS1	"xxx"
@ADDRESS2	

Figure 23 Translate Fixed Record File Format Into XML Format

In the above implementation, a large fixed records file is read in by the Parse Data activity. The outer iteration group allows the Parse Data activity to read only 100 records at a time until EOF. The Render XML activity translate all the records read in by the Parse Data activity into XML document and accumulate the result in a list, which is passed to the sub-process Process XML Record for further processing. This way, the memory consumption of the BW process engine can be controlled to a reasonable level.

Applicability

XML is string based, thus supported on all platforms. We recommend that all new internal applications use XML as its standard data format.

To integrate existing applications, we need to translate their data into the standard XML format.

Related Patterns

Standard Wire Format, Canonical Data Schema

4.2 Standard Wire Format

Context

During enterprise application integration (EAI), messaging middleware is used to transfer data among the integrated applications.

RV has a self-describing data type system, which supports a variety of different data types. For instance, XML is one of the types natively supported by RV. For a complete list of data types supported by RV, please refer to *TIBCO Rendezvous Concepts, Software Release 7.0, Chapter 6 Data* for details.

In case of EMS we have rich set of message types as well. This includes Text (this could be XML message), Map (value pairs), Bytes, Stream and Object messages. Complete description can be found in *TIBCO EMS User's Guide*.

Applications can use RV API or JMS API to create self-describing data structure that is used by RV or JMS correspondingly to carry application data payload. The self-describing data structure is a list of name-value pairs, the type of which can be any supported RV data type in case of RV. This self-describing data structure used by RV to represent application data payload is called the data's *wire format*. In general, wire format is the data structure used by the messaging middleware to represent application data payload. In case of JMS we have greater variety of self-describing data as text, map, bytes, stream and object.

Problem

If different applications using RV or JMS to transfer data use different wire format, the senders and receivers of data cannot understand each other, as the receivers of data do not necessarily know what field to extract from the RV or JMS message.

Solution

The following figure illustrates a publisher and a subscriber, each has its own data format. To publish data from the publisher to the subscriber, the publisher must translate its data from "data format 1" into "wire format 1" before it puts the data on the message bus, and the subscriber must translate the data received from the message bus from the wire format ("wire format 2") into "data format 2". If the two wire formats are different, one must be converted to the other in order for applications to parse the data.

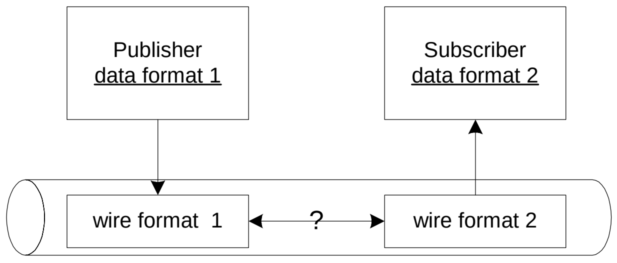


Figure 24 Using Wire Formats

In case of using RV as transport, rather than allowing applications to use many different wire formats, a standard wire format will solve the incompatibility issue and is easily enforceable. The current version of TIBCO ActiveEnterprise (AE) adapters and TIBCO BusinessWorks support two RV wire formats:

- ***rvMsg*** wire format is of data type **TIBRVMSG_MSG**. AE adapters do not perform automatic validation on data with this format, but if you use this format, AE adapters are compatible with custom-built, non-SDK based adapters.
- ***aeRvMsg*** wire format is the TIBCO ActiveEnterprise standard wire format. It is also of data type **TIBRVMSG_MSG**, but provides class information and packing rules for the TIBCO Adapter SDK set of data types (supported by SDK based TIBCO adapters and BW). This format allows ActiveEnterprise components to perform extra validation on messages sent or received. Data represented in aeRvMsg format is called *AE objects*.

Note that aeRvMsg is also referred to as "AE Msg" or "AEMsg" by many people so these terms are used interchangeably in this document.

Standard wire formatting in JMS case involves encoding JMS message payload as XML and defining header and property fields as described in *TIBCO EMS User's Guide*.

Applicability

We recommend all internal applications that use JMS to integrate with other applications use TextMessage type for JMS payload, encoded as XML.

Related Patterns

Canonical Data Schema, Bridge, Standard Data Format

4.3 Canonical Data Schema

Context

When an application is integrated with another application, it needs to know how to parse the data coming from the other application, in addition to knowing the data format. Most modern software systems support introspection of its data structure of unknown (or ANY) type. Introspection is a programmatic approach to find out the semantics of the data without prior knowledge of how the data are constructed. For example, Java or COM can use its reflection API to introspect an object of ANY type; RV API also provides similar functions on RV messages.

***Data schema* is the semantic representation of data, and it is also called *metadata*.** Each software system describe data schema differently. For example, XML can use DTD or XSD to represent XML schema. A customer object represented in XML format cannot be parsed without prior knowledge of its schema information, because it may have one address line, or it may have two address lines. Another example is RDBMS such as MS SQL Server. Relational database represents data schema in database schema.

Problem

Although most modern software systems provide introspection function, it is not desirable to use it in practice. Ideally, data schema of a data object is known in advance. However, there are two issues around using data schemas.

One issue is how to describe and share data schemas. Programming languages depend on language features to define and persist schemas. XML, on the other hand, uses standard based schema specification such as DTD or XSD. Both DTD and XSD are ASCII based and can be easily transferred and shared among applications.

Even if every application in the world uses the same schema specification, say, XSD, we still have another issue. For example, the Customer object in SAP is defined differently from the Customer object in Siebel, even if they both refer to customer. The SAP Customer could store Customer's name in a XML element called "CUST_NAME", whereas Siebel could store Customer's name in a XML element called "CustomerName". Obviously, in order for the two applications to share the customer object with each other, data represented in one data schema must be converted into data using another schema. **The process of converting data from one data schema to the other is called *data transformation***.

When more than three systems that need to be integrated have different data schemas, generating transformations between each pair escalates into an O(n^2) can be a problem. The total number of transformations that one could build is n*(n-1) for n systems with disparate data schemas.

Solution

The solution is to build canonical data schema (the word canonical means conforming to a general rule or acceptable procedure) to represent the data model, and transform data in native application data schema into data in canonical data schema. The total number of transformations that one needs to build is only 2*n for n systems with disparate data schemas. The total number of data transformations can be reduced to O(n).

Every application should transform data in native application schema into data in canonical data schema, if not already representing data in the canonical form. Theoretically, this only makes sense when there are more than two applications that need to mutually share information, and the number of applications is greater than three. The following diagram illustrates this idea:

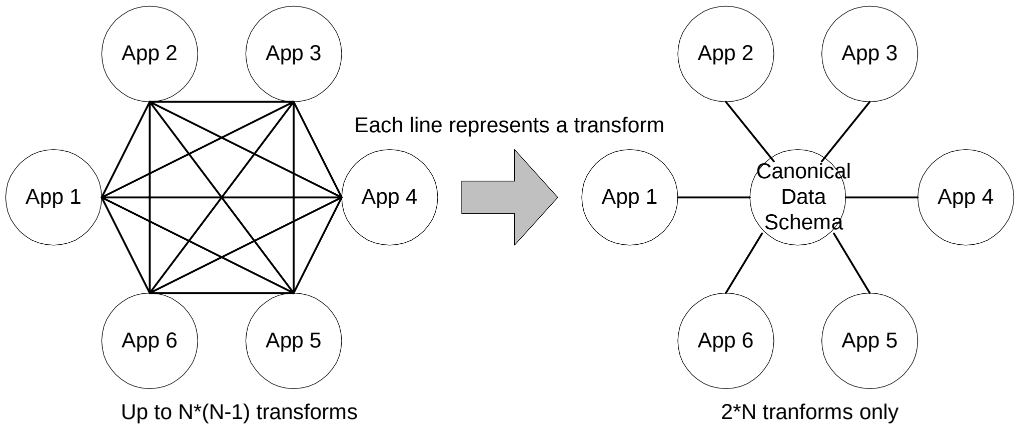


Figure 25 Canonical Data Schema

One of the industry standards to represent data schema for XML data is XSD (XML Schema Definition). XSD has a more comprehensive type system than DTD, thus more suitable for building strong typed systems. If XML has been chosen as the standard data format, XSD is the natural choice as the canonical data schema. Using canonical data schema, we can achieve the following benefits:

- Data schemas are decoupled from each other among applications. When the data schema changes in one application, we only need to update the transformations that convert between the application native schema and the canonical data schema without affecting others.
- We can achieve reuse of the data transformations and better maintainability in the long run.
- This is a scalable and extensible approach to build transformations.

XSD can be created at design time using such authoring tools such as TIBCO XML Authority (part of TIBCO Extensibility tool suite). Parse XML activity can be used to validate and parse XML documents into internal XML schema trees in BW, which can be mapped or transformed using XPATH expressions or the Mapper activity. To convert internal XML schema trees in BW back into XML documents, use the Render XML activity.

Existing applications can continue to use their native data schema, but will need to be transformed into the canonical data schema using the *Transformer* pattern (see next section 3.4) if application integration is required.

Applicability
We recommend using canonical data schema to represent data in new applications that need to communicate with other internal applications in the intranet environment, whether the applications are intra-office applications or inter-office applications.

Defining canonical data schema requires complete understanding of business functions and data models used in those business functions. Coordination between owners of different applications is inevitable to make this effort a success.

Related Patterns
Transformer, Standard Data Format

4.4 Transformer

Context

Different applications, whether homegrown or from different vendors, can use different data schema to represent even the same data. For instance, one application can name a customer attribute “address”, whereas the other may use “addr” to represent the same address. In some other scenarios, one application may need to aggregate data from several applications or extract subsets of data from others. All the above scenarios add complexity to integrating disparate applications, yet they occur very often in real world.

Problem

Data schemas could be different because they could contain different number of fields, and the corresponding fields can have different types or names, or etc. Defining canonical data schemas can simply this issue (see section 3.3). When an application does not represent data in the canonical data schema, it must convert its data from the native schema to data in the canonical form.

Solution

In a broader context, transformer can be used to convert data from one data schema to data in another schema (see next figure).

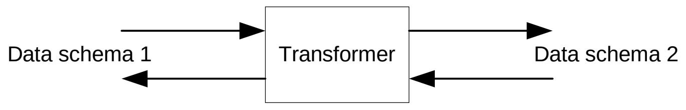


Figure 26 Transformer Pattern

In our context, the transformer pattern is used to convert data from its native schema to data in the canonical form, or vice versa. All applications understand canonical data schema and can use it to parse data they receive.

Implementation

TIBCO BusinessWorks is the primary platform for data transformation. Normally, transformation of data occurs by mapping schemas of input process variables to an activity’s input schema. If you wish to change the schema of input process variables before it gets placed into the activity input schema, you can use any valid XPath expression to do the transformation. You can type in the XPath expression into the activity input item’s expression field, or you can click the XPath formula builder button to create an XPath expression.

In addition, the Mapper activity is used to map input process variables to a new process variable defined by the activity’s Output Schema, also using XPath expressions.

The following figure illustrates how input process variable schemas can be transformed (mapped) into the input schema of an activity:

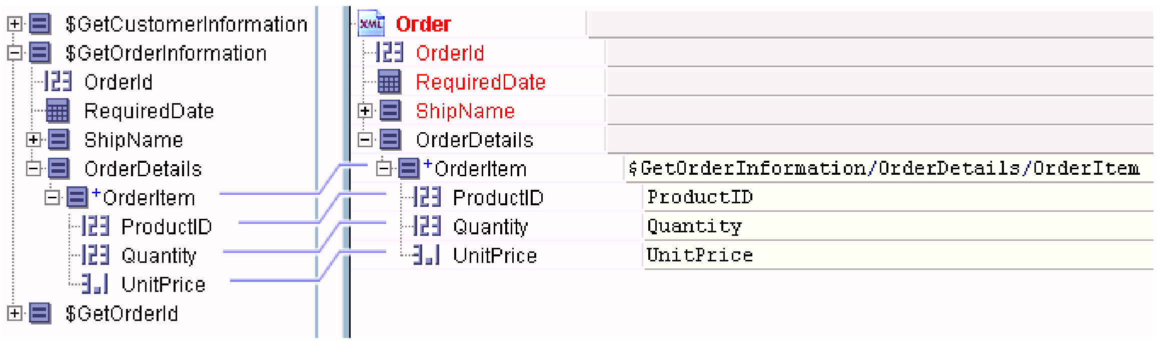


Figure 27 Transforming Input Process Variables (Left) to Activity Input (Right)

The input schema of an activity is represented internally as Extensible Stylesheet Language Transformation (XSLT) code. To reuse the XLST code, you can right-click on any item in the input schema and choose Copy from the popup menu. Then open a blank text document and choose Paste. The XSLT is displayed in your text document.

The above approach to transform data is best suited for developers and integration specialists with little background in XSLT, as it provides easy-to-use drag and drop functionality. It does require, however, the person who use this BW feature to understand some basic XPATH syntax.

Another scenario when it is necessary to use this approach is when the schemas to transform from or to are not in XML schema format. For example, to map a flat file schema, a relational database schema, or an ActiveEnterprise (AE) schema (all schemas defined in the Schema folder in your BW project are AE schemas) to a XML schema and vice versa, you need to use this approach.

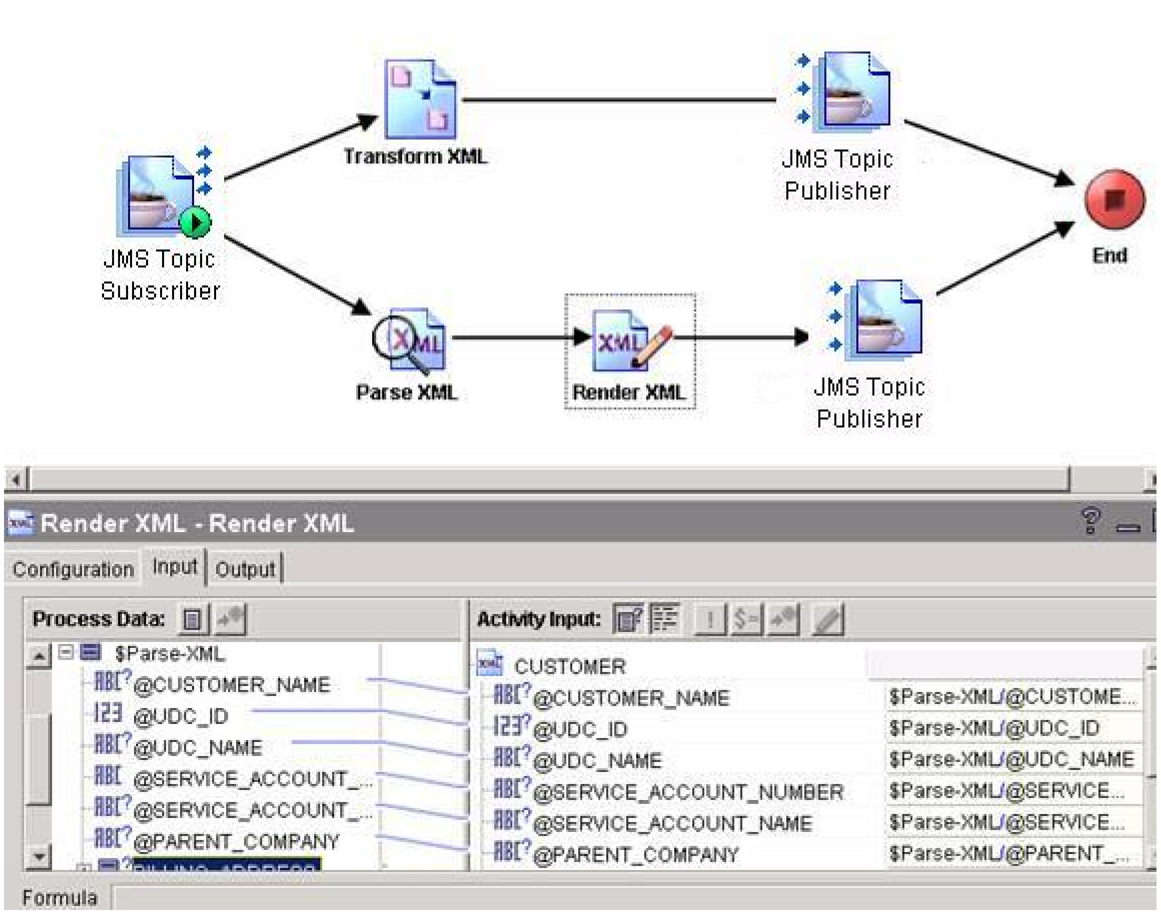
Using XSLT Directly

If you know how to build a XSLT file that are used for transformations, or if you are supplied with an XSLT file from an outside source, the Transform XML activity allows you to use the XSLT file instead of manually creating the mappings.

Performance Consideration

The two different approaches implemented in BW are illustrated in the same figure on the following page. The JMS subscriber subscribes a message from the wire. To transform the XML document contained in the incoming message into the canonical schema “Customer”, the top branch of the BW process uses XSLT directly, whereas the lower branch parses the XML document into an internal schema tree, and maps it to the input of Render XML activity.

The bottom branch consumes more system memory and takes more CPU time to complete the transformation. Consequently, when performance is a key consideration, we recommend using XSLT to do data transformation directly.



Process Data	Activity Input	
$Parse-XML	CUSTOMER	
@CUSTOMER_NAME	@CUSTOMER_NAME	$Parse-XML/@CUSTOME...
@UDC_ID	@UDC_ID	$Parse-XML/@UDC_ID
@UDC_NAME	@UDC_NAME	$Parse-XML/@UDC_NAME
@SERVICE_ACCOUNT_...	@SERVICE_ACCOUNT_NUMBER	$Parse-XML/@SERVICE...
@SERVICE_ACCOUNT_...	@SERVICE_ACCOUNT_NAME	$Parse-XML/@SERVICE...
@PARENT_COMPANY	@PARENT_COMPANY	$Parse-XML/@PARENT_...

Figure 28 Two Different Approaches to Transform Data

Applicability

Use the transformer pattern in all applications when the native application data schema is not in canonical data schema form, and the application needs to integrate with other applications.

Related Patterns

Canonical Data Schema

4.5 Data Validation

Context

One of the most basic tasks in Enterprise Application Integration (EAI) is sharing data. Data could contain errors that are generated by humans, applications or during transmission. Data validation is the process of identifying data errors using such constraints defined in data schemas as type, range, cardinality (zero or more occurrence), and etc.

Problem

Unexpected data errors could crash applications. Even worse, when data errors are allowed to travel from the applications where they occur to downstream applications where they are consumed, the downstream applications usually have no means of correcting and reporting the errors because the data are already out of context. So it is desirable to catch data errors and catch them early.

If data errors could be caught early, they would not waste valuable system resources traveling to downstream components, and errors could be corrected as early as possible.

Solution

Since most data will be represented in XML, an XSD associated with the data can be used to validate the data as soon as the data is created. In a publish-subscribe scenario, data should always be validated in the publisher application, even if the consumer of the data is the subscriber. The rule of thumb of data validation is to validate them at the earliest possible stage.

In BusinessWorks, the Parse XML activity is used to parse the XML data. If the XML data exists as files, the preferred way to parse XML files is to use a Read File activity set to binary mode to read the XML file, and then pass the binary file contents to the Parse XML activity.

Data errors caught should be passed along to the generic exception handling process for logging and reporting.

Applicability

In practice, we recommend exercising data validation at the beginning of each BW process as long as that process has incoming data (such as JMS/RV Subscriber, File Reader, and etc.), or right after the activity where data are created.

Related Patterns

Standard Data Format, Exception Handling Process

5 Process Management Patterns

Process management patterns deal with design patterns in business process management (BPM) applications, they provide process design and implementation techniques in the integration broker and workflow manager based development environment. The following discussions use BusinessWorks (BW) as the standard integration broker used to develop integration applications.

5.1 Integration Broker

Context

Most businesses choose the best-of-breed applications in their environment for different core business functions. For example, a company can use a variety of vendor-supplied applications to do order-entry, catalog, credit checking, inventory management, order management, and shipping for e-commerce. Each application is critical, but the flow of information and processing between each application is what drives the day-to-day operations of the business. For example, the order-entry application that receives and processes orders based on availability information taken from the inventory system. The tracking system relies on order information and shipping information. Accurate and up-to-date information is critical for the efficient running of the business.

Problem

Traditionally, business rules that tie these systems together are provided by custom-written code. For example, application servers provide a programming environment for developers to write “glue” code that ties together different enterprise backend systems. However, tying together different systems from different vendors that run in different environments is a labor-intensive and error-prone process that usually takes a long time to plan and implement. In addition, the task of building business logic is very complex by itself (one has to deal with process synchronization, concurrency management, state management, etc.). In fact, many businesses still rely on manual processes that have many data entry points instead of automating the process for greater efficiency and accuracy.

Solution

If business process were a symphony, then different applications participating in the business process were performers such as pianists and violinists, and Integration Broker is the conductor that orchestrates the entire business process. Integration Broker is mainly used to tie together different systems as participants of a business process and automate the processing of business logic via pre-defined business rules. This allows you to reduce the time to implement integration applications and ultimately lower the cost of deploying and maintaining such systems.

Integration Broker usually consists of a design environment in which you use to model business processes without involving much custom coding, and a scalable runtime environment in which the defined business processes will be executed.

Implementation

TIBCO BusinessWorks (BW) is the tool that implements the Integration Broker pattern. BW has a graphical Designer that includes a long list of pre-built tasks with drag and drop easy-of-use to model processes. In addition, you can use BW’s programming environment (to write scripted tasks or custom tasks in Java) to code more complex business logic if necessary. Once you defined the business rules using BW’s Designer environment, BW Engines can execute the modeled business processes at runtime, allowing you to easily integrate and automate the flow of your business.

The current version of BW models business processes after UML's activity diagram. UML activity diagram consists of modeling elements called "task" and "trigger". Task is a unit of business logic that constitutes a "process". It can be used to model the interaction with an external system such as database query, SAP BAPI call, or updating the internal state of the current business process. Trigger, also known as transition, is used to flow execution from one task to another. Together, tasks and triggers form a well defined business process. BW's process definition is saved as XML documents (TIBCO proprietary schema) in TIBCO Repository (a sub process of the TIBCO Administration Server). Job, which is a runtime instance of a BW process, can be triggered by an external event and created by the BW Engine based on predefined rules. The following topics discuss different design considerations when using the Integration Broker pattern.

Integration Broker must maintain some state information shared between systems participating in the business process. In TIBCO's implementation of BW, a process's life cycle is limited to that of a job, which is a runtime thread. By default, the process state information is kept only in memory without being persisted. There is advantage and disadvantage associated with this implementation. The advantage is that the performance is fast; the disadvantage is that when the BW engine crashes in the middle of a running job, the state cannot be recovered. BW provides a "checkpoint" task to help with persist the entire state of the running process. It's like taking a snapshot of the running process and saving it into non-volatile storage. When system crashes and restarts, BW will be able to pick up the process state from its previous checkpoint and restart from there.

<u>Anti-patterns</u>

- Integration Broker cannot manage a business process that requires human involvement, such as human approval and tasks that can only be done by a human (such as shipping a product).
- Integration Broker cannot manage a business process that requires dynamic changes (of process models) at runtime.
- BW keeps process state information in memory. Memory is still considered an expensive system resource compared with disk space. Unless state information can be inherently persisted (which is not a case in BW), we do not recommend using BW to manage business processes whose life cycle span over a long period of time.
- Integration Broker is used primarily to integrate with existing business functions rather than creating them. There is no sophisticated development environment associated with TIBCO BW. As a result, we recommend against developing too much scripted business logic into the Integration Broker itself. Instead, if certain business logic must be done, write it in Java and include as a custom Java task in the process.

<u>Application Server vs. Integration Broker</u>

It has been argued that application server can be used as Integration Broker or to implement the Integration Broker pattern that ties all different systems together. However, the skills needed to develop business processes via custom coding such as java programming or EJB are high and demanding. The main challenge is how to coordinate multiple participants in a business process (this can be a database, a legacy app, or a CRM or ERP system) and synchronize one with the other. The other challenge is to how build a scalable system that can handle multiple instances of the business process to run concurrently. Using Integration Broker instead can replace the application server session bean logic that is used to manage the workflow of a business process, resulting in a much more scalable integrated system.

Consider a simplified business process of placing orders. The business process of placing orders involves three steps. The first step is to check whether the total amount of the order has exceeded the credit limit. If no, the second step checks for inventory. If the inventory has enough orders, the third step is to place orders by updating the order database. After all three steps are successful, the order status is returned to the client. What is not simple in this simplified business process is that both the first and second steps require an external communication to legacy financial and inventory systems. The following figure illustrates a typical implementation using EJB app server.

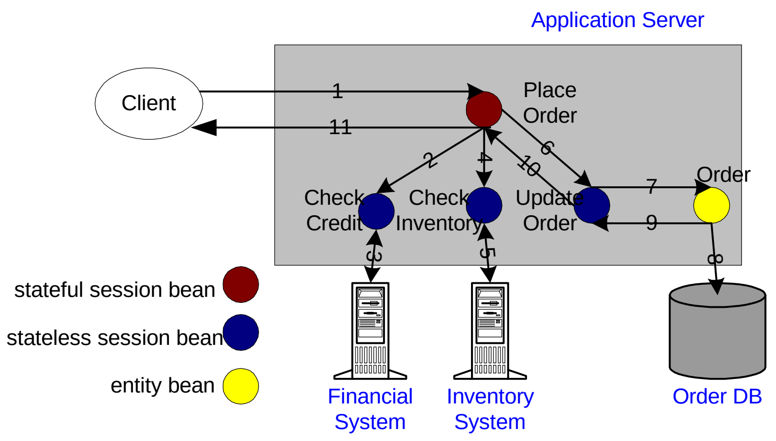


Figure 29 Place Order Using EJB App Server

In the above figure, the stateful session bean PlaceOrder maintains the conversational state for the business process initiated by the client. The numbers on the arrowed lines indicate the sequence of this process.

Let’s now take a look at how to use the Integration Broker system pattern to solve this problem. TIBCO BusinessWorks (BW) is used as the platform to implement this process. A person without much programming experience can use the BW Designer GUI interface to implement the same “place order” business process in no time. The next figure shows this different integration approach. BW is the execution engine of the business process in this design.

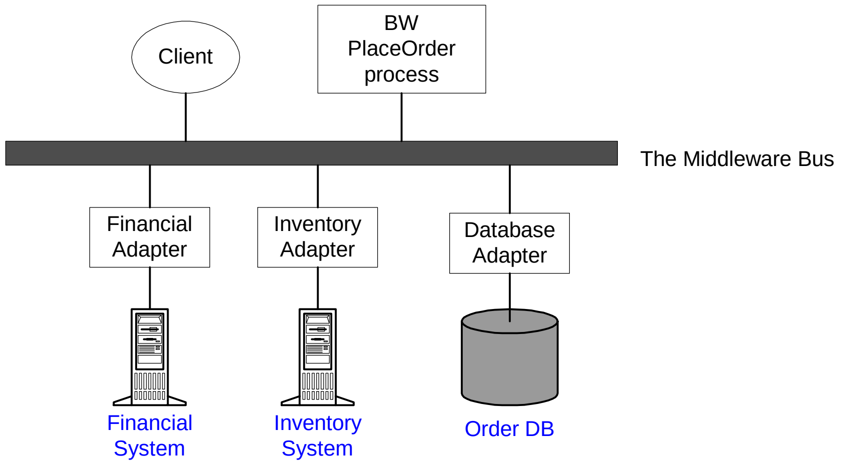


Figure 30 Place Order Using Business Process Automation

The next figure shows the BW process diagram for the “place order” business process.

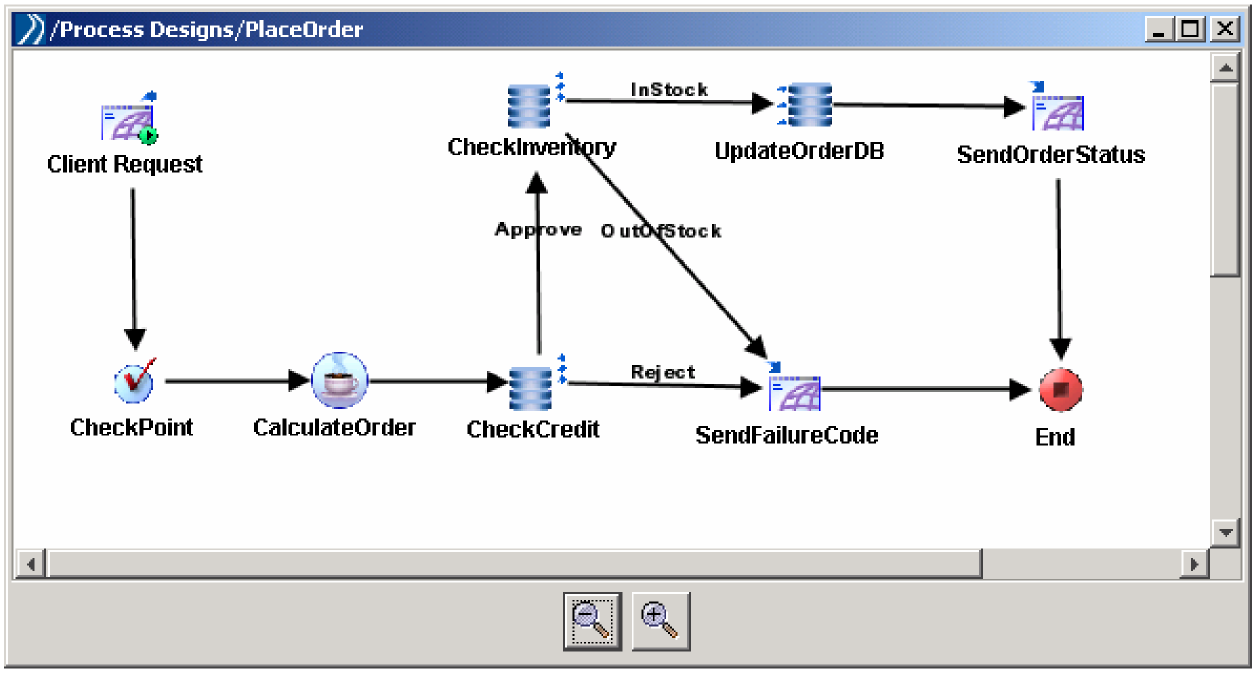


Figure 31 Place Order Process Diagram in BW

Like application servers, BW also provides a multi-threaded host environment for executing jobs (instances of processes) at runtime. BW deployment configurations can pool a configurable number of jobs similar to the way that app server pools EJBs in its containers. BW also provides transaction management, dynamic fail over and load balancing schemes (refer to the Load Balancer system pattern).

The benefits that BW has over app servers include the following:

- Intuitive GUI based business process design environment that is based on UML activity diagram. Without much Java programming background, process designers can focus on business process level design in a WYSIWYG fashion.
- Enterprise Java Beans are all single threaded Java object. This is not always efficient when the business process contains multiple tasks (sub-processes) that can be executed concurrently. EJB applications cannot exploit the concurrency and have to execute the logic sequentially. BW can model this concurrency using its “sync-bar” task, providing maximum concurrency.
- BW keeps state information in process memory. It can take snapshots of the entire state of the running process at any point and persist that state onto the disk. This is especially useful for long-running processes (which involve multi-step communications to different backend systems). In app server, the state information is kept in stateful session beans. When the server crashes, all previous work is lost and the client has to start over from making a new request. When BW servers crash, the BW engine can pick up the saved state information from the previous checkpoint rather than starting over from the beginning.
- BW supports many different styles of messaging (synchronous or asynchronous) protocols natively. This includes JMS, RV, HTTP, HTTPS, FTP, CORBA, and Java RMI.
- BW also provides many convenient tasks such as those interfacing with the databases and file systems.
- However, BW does not itself provide any security management or distributed transaction management. App servers’ main strength is that it provides a highly flexible programming environment for developing custom

business logic, but app servers do not (or at least not yet) provide an easy environment to do business process design as intuitively as those offered by Integration Brokers products such as BW based on the above discussion.

Applicability

Using Integration Broker must satisfy the following prerequisites:

- The business process can be fully automated, without any need of human intervention. (Note: It is not surprising to see that certain Integration Broker implementations such as BusinessWorks will include human task plug-ins such as InConcert plug-in to manage human activities. Nevertheless, Integration Broker remains its focus on process automation).
- The entire business process can be statically defined at design time. In other words, there are no unanticipated business events that can occur at runtime.
- It can be used to get rid of the traditional batch processes and implement the straight-through processing (STP).
- Integration Broker is targeted at technical developers who understand APIs, data formats, data schemas, messaging technology, and scripting. It is not meant for business process modelers or business analysts who are more interested in business logic than its technical implementations. Refer to Workflow Manager pattern for more details.
- At last, the following table compares integration brokers with application servers with the use cases/applications suitable for each under the respective column:

Integration Broker	**Application Server**
Data access and query (incl. request and reply)	Session management
Access legacy systems (such as CMS, LMS, EMS, PDB)	Distributed transaction
Data transformation	Java programming
Message-oriented applications	Object-oriented programming
Posting or publishing data	

Table 10 Compare Integration Broker and Application Server

Related Patterns

Workflow Manager

5.2 Workflow Manager

Context

A complex business process not only relies on non-human applications, but also on human to act on important business events. For example, many business processes must include human signoffs and approvals as part of the business requirements for auditing purposes. In addition, business managers need to gain a high degree of visibility into business processes to track and measure performance for improvements.

Problem

Integration Broker pattern solves the business process automation problem. However, when a business process must involve human or involve dynamic changes, Integration Broker is not up to the task. Because of the complexity of integrating human activities as part of the automated business process, most steps that involve human are still handled manually and this process is not efficient and is error-prone.

Solution

Because business process that involve human can take a long time to complete, the state of the process must be stored in persistent storage such as disks. It must provide a single business process management solution that can integrate not only human elements, but also non-human elements such as external systems, databases, etc. into the business process seamlessly. The Workflow Manager pattern must offer an easy-to-use design environment and a runtime environment that scales on both system load as well as unanticipated business logic changes.

Workflow Manager can provide the following benefits to your business:

- Streamline operations
- Achieve regulatory compliances
- Provide process performance metrics, gaining high degree of visibility
- Bring products to market faster
- Improve customer service/satisfaction

Implementation

TIBCO InConcert (IC) implements the Workflow Manager pattern. TIBCO IC consists of process design environment (PC GUI) and runtime environment includes IC servers, IC Repository, and IC Rules Agents.

IC processes consist of tasks and transitions between tasks. To model the human elements into the business process, each IC task has a role. Each role must be assigned to a pool of users. Any users within the user pool can acquire the task on a first-come-first-serve basis. Each task is associated with one of the many states. A task will be in Waiting state when it is not ready to be executed. When a task's state is Ready, it can be acquired by a user and become the Active state. When the task is completed, the user can complete the task. The user can also skip the task entirely without processing it. In either case, the task's state will become Done, and the transition will happen and change the state of the next task in the process from Waiting to Ready. Because the flow of the process is controlled by setting the state of each task, both human and non-human elements can participate in the process by doing that either manually (in the case of humans) or automatically (through writing custom code in IC API). The following graphic represents the states of a task in an IC workflow and all possible transitions between the states.

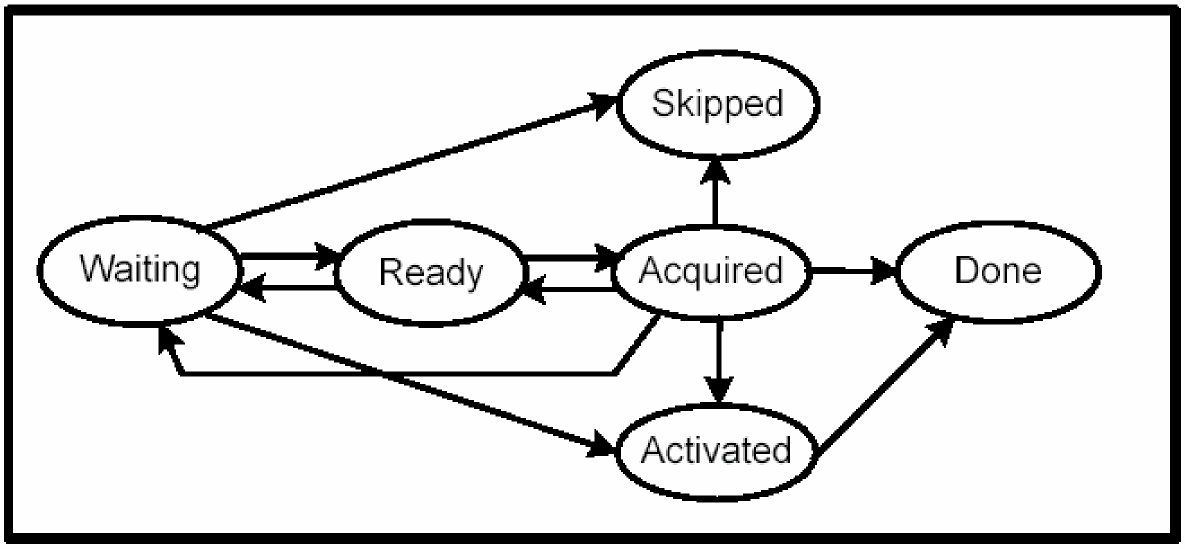


Figure 32 IC Task State Transitions

In addition, IC's process definition and state information are all kept in IC database. The IC server is the gateway from which this information can be accessed. The result of every single operation on the task that changes its state is saved into the IC database. As a result, the performance of IC is quite I/O dependent and should be carefully designed to avoid unnecessary performance penalties.

A common pitfall of using IC is that one tends to store a lot of state information with the IC process (by defining process or task level attributes) and carry it between tasks. Unlike BW, this is NOT a good practice because ALL state information will be automatically persisted into the IC database at every single step. This can become prohibitively expensive when you carry the state throughout your IC process and the IC process is composed of hundreds or more tasks. The performance will be extremely slow and not scalable. We always recommend keeping as little state information as possible within IC.

Compare BW and IC

IC provides state-based workflow that is suited for business processes that are long-lived, dynamic, and involve human activities. Whereas BW provides event-based process automation suitable for complex business processes that are short-lived, defined and static, and require high-throughput. The following table compares the major differences:

TIBCO InConcert (Workflow Manager)	**TIBCO BusinessWorks (Integration Broker)**
Good at managing long lived processes	Not good at managing long lived processes: will accumulate too many waiting processes which will lead to performance problems
Not good at managing high volume of short lived processes: too much overhead	Good at managing high volume of short lived processes
Accounts for each process step / task in the database, provides increased visibility and statistics	No such features built in; would create too much overhead for short lived processes
Roles and groups and users management	No notion of users and user groups and the respective statuses of a task life cycle
Can remodel process changes dynamically	Does not have this concept

Triggering events based on human activities, such as task completion, expiration, etc.	Triggering events based on non-human, system events such as incoming messages, timers, and file system changes

Table 11 BW and IC Comparison

Not only can IC be used to model business processes that involves human activities (which is what IC is the best at), IC can also manager/include non-human elements in the business process. The human element can be inherently managed by the way that IC works because all state is persisted at each and every single step. IC agents can execute the non-human element. The IC agent can be implemented by writing custom code using IC APIs (in C++ or Java), using or extending the pre-defined IC Rules Agents (by writing Prolog based agent rules), or, most suitably, invoking an automated business process that is managed by the Integration Broker!

The last option of implement an IC agent requires the integration of IC and BW. The integration between IC and BW was achieved in TIBCO's BW product through the use of Manual Tasks. TIBCO BW Designer is a design time GUI combining pre-defined IC and BW tasks. Key features of the BW Manual Task Interface include:

- Common design environment for workflow and process automation
- Drag-and-drop workflows and process modeling
- Data mapping between IC tasks and BW processes
- Changes are propagated in IC automatically
- Ability to start processes based on IC task state or automatic business events
- Run-time coordination among IC and BW using internal bridges

Anti-patterns

Many pre-packaged vendor applications provide domain specific workflow solutions, such as Vantive workflow used for customer care. The Workflow Manager discussed above is a generic workflow manager than is highly adaptive to different business requirements. However, it is not meant to replace existing workflow solutions that do its work well.

Applicability

Using the Workflow Manager pattern when:

- The business process involve human activates.
- The workflow tasks should be only assigned to users of certain roles.
- The business process could change dynamically at runtime to anticipate unforeseen business events.
- Workflow Manager is targeted at workflow developers that understand modeled business processes and business objects that associate with the process.

Together with the Integration Broker (IB) pattern, the Workflow Manager (WFM) pattern can be used to form a number of business process management reference designs.

Related Patterns

Integration Broker

5.3 Process/Task Delegation

Context

This pattern is used in process implementation in the BW Designer environment.

Problem

Some business processes implemented in BW Designer can consist of hundreds of activities, if not more. It is difficult for others to read or maintain such a big process.

Solution

In procedural programming languages such as Java and C++, we promote the use of subroutines or functions as a good practice of modular design. Similarly, in implementing business processes, we promote the use of sub-processes so that each process is concise and readable, thus easier to read and maintain. The following figure conceptually illustrates this idea: the process consisted of task 3, 4, and 5 is delegated from task 2 in the parent process (1, 2, and 3).

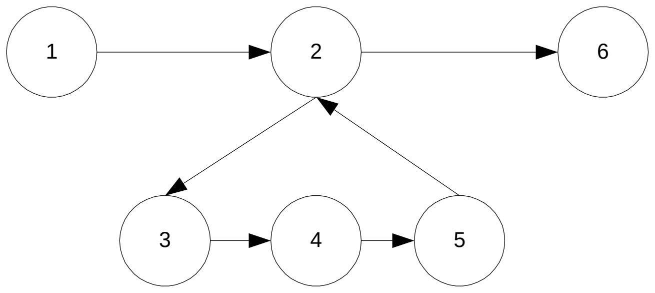


Figure 33 Process or Task Delegation

The Call Process activity can be used in BW Designer for this purpose. Calling a subprocess is similar to calling a subroutine in a procedural programming language. Normally, a subprocess executes within the same job thread as the calling process, and the output of the subprocess is available to all subsequent activities in the process. If the checkbox ('Spawn') in the configuration tab of the Call Process activity is checked, a new process is spawned for the subprocess. When a subprocess spawns a new process instance, the parent process cannot access the called process' output. The checkbox should therefore be left unchecked for this pattern.

This pattern is also known as *Task Delegation*.

Implementation

The figure below shows an implementation of this pattern. The parent process shown in this figure receives requests for a business function processing via HTTP. It delegates the task of processing the request a child process. The results obtained from the child process are sent back to the requester via HTTP.

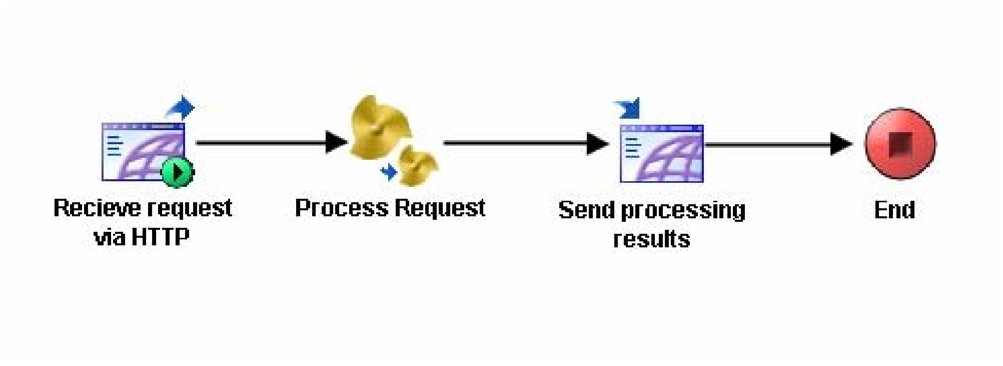


Figure 34 Process Delegation Example

Applicability

Use this pattern to promote modular process design when the integrated business process consists of too many tasks (that cannot be cleanly fit into one page without scrolling).

Related Patterns

Parallel Execution

5.4 Process Synchronization

Context

You may need process instances (i.e. jobs) to communicate if you wish to synchronize process execution or if your processes must execute in a specific order.

Problem

In BW, jobs are executed in separate threads in the same BW engine or different engines if multiple BW engines are deployed. To communicate, jobs can send messages between each other. This pattern explores different ways of how jobs communicate within this context.

Solution

One way is to use general RV activities to send and receive messages between each other. Send RV Request activity will block until the request comes back from the other process.

The other way is to use the Wait and Notify activities provided by BW. These activities are similar to semaphores in programming. A process containing a Wait activity waits for another process to execute a corresponding Notify activity. Alternatively, the Receive Notification process starter creates a new process instance when another process executes the corresponding Notify activity. This is easier than using direct messaging such as RV between running jobs.

Implementation

The following example illustrates an implementation of this pattern using a order processing scenario - a new order arrives, and because the total is over $100,000, it requires approval before processing. You may notify a group of approval managers by email, and then any of the approval managers can respond by email or by way of a web interface for approval.

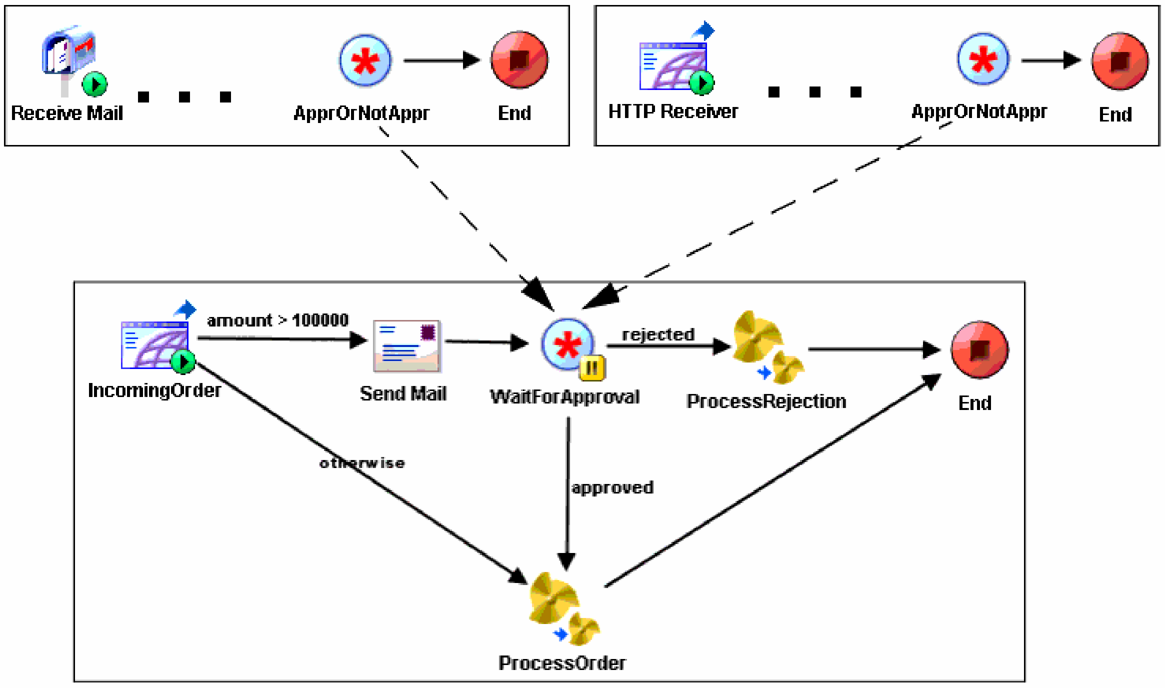


Figure 35 Process Synchronization Example

Applicability

Use this system pattern to support process synchronization.

Related Patterns

Parallel Execution

5.5 Parallel Execution

Context

This pattern is used in the context of implementing business processes using BW Designer interface.

Problem

In a single linear control flow, tasks are executed in sequence and each task is dependent on the preceding task. One task that communicates with an external system, such as requesting from a remote server or executing a lengthy database transaction, could create a bottleneck in the process flow. Meanwhile, other tasks are able to continue processing without waiting for the completion of this task.

Solution

To create a new job to execute the parallel tasks in a new process flow, one can use the Call Process activity to spawn a new job in one of the split process flows. The separate job created by Call Process task is called the *called process*, or *subprocess*.

Implementation

In BW, the subprocess will be executed in parallel (in a new job) if the Spawn field of the configuration tab of the Call Process activity is checked. In order for the parent process cannot access the called process' output, the parent process and the subprocess must use process synchronization techniques discussed in the previous pattern.

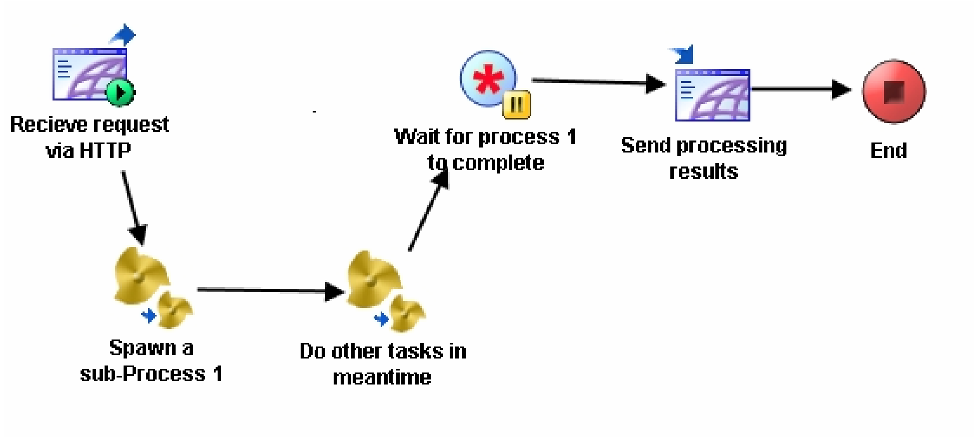


Figure 36 Parallel Execution Using Wait and Notify activities in BW

Applicability

Use the parallel execution pattern to split processes into multiple flows when both of the following conditions are true:

- One task in the process flow could create a bottleneck, such as waiting for an incoming event or a long external request operation.
- There are other tasks that can execute in parallel.

Related Patterns

Process Synchronization, Process/Task Delegation

5.6 Sequential Execution

Context
Sometimes business logic requires incoming events to be processed sequentially.

Problem
BW engine is a multithreaded model that allows processing of incoming events in parallel.

Solution
Process instances can be executed in the order they were created. TIBCO BusinessWorks (5.2) allows you to specify a sequencing key on process starters that determines which process instances to execute sequentially. Process instances with sequencing keys that evaluate to the same value are executed in the order they were started. Process instances can have sequencing keys that evaluate to different values, but only process instances with the same value for the sequencing key are executed sequentially. Process instances with different sequencing key values can be executed concurrently.

Alternatively, BW Process Engine deployment resource allows you to specify the maximum number of process instances that can concurrently be loaded into memory by configuring the Max Job parameter. In addition, by configuring the Activation Limit parameter, you can specify that once a process instance is loaded, it must remain in memory until it completes.

To ensure sequential processing of incoming events, set Max Job to 1, and Activation Limit to enabled. As an event arrives, the process engine creates a process instance and keeps it in memory. The maximum number of process instances in memory is one, so only one process can execute at a time because the currently loaded process cannot leave memory until it completes. The process engine creates process instances for new incoming events, but pages them to disk immediately until the first process instance completes. Once the process instance completes, the process instance for the second event is processed, and so on.

The following figure illustrates an implementation of this pattern. In this illustration, an invoicing business process can receive multiple requests in very short period of time. Since the process involves a credit balance check and update, sequential processing of the incoming requests is important.

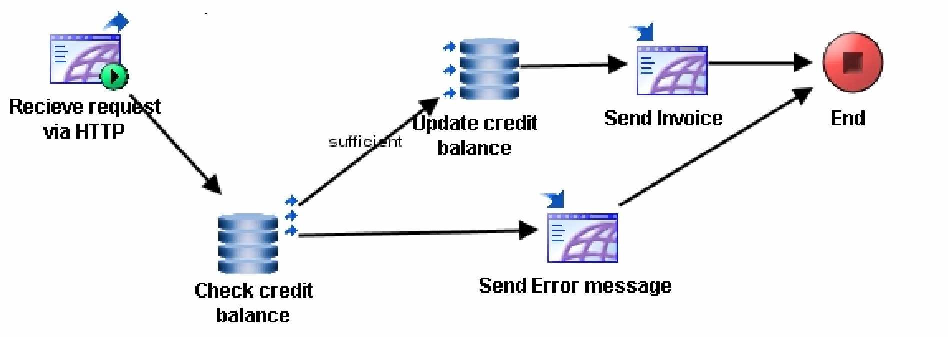


Figure 37Sequencing Process Instances

By setting the ‘Max Job’ property of this process to 1, all incoming requests will be processed sequentially, ensuring that the credit balance check and updates are correct.

Applicability

Use this configuration when you must ensure sequential processing of incoming events.

Related Patterns

None

5.7 Shared Properties

Context

Jobs executed in Integration Broker engines need to access property settings at runtime, for instance, a logging flag that can be set dynamically to trace application status at different log level.

Problem

Normally, properties are defined statically in global variables or property files and cannot be changed at runtime. However, we need a mechanism to update property values (whether by external programs or internally by running jobs) dynamically at runtime, and make the updated property values accessible by any job.

Solution

In BW 5.2, a Shared Variable resource allows you to share data across process instances. All process instances can read and update the data stored in a shared variable. This type of shared variable is useful if you wish to pass data across process instances or if you wish to make a common set of information available to all process instances.

Because multiple process instances can potentially access and assign values to Shared Variable resources, the Lock shared configuration object and critical section group allow you to synchronize access to Shared Variable resources. Without a mechanism for locking, a process instance could update the value of a variable while another process instance is attempting to read the value. This would result in an unpredictable value for the variable.

You should use critical section groups to contain the Set Shared Variable and Get Shared Variable activities. This ensures that only one process instance attempts to assign a value to the variable and ensures that no process assigns a value to the variable when the current process attempts to read the value.

Alternatively, we can use BW to build a property cache process. This is illustrated in the following figure:

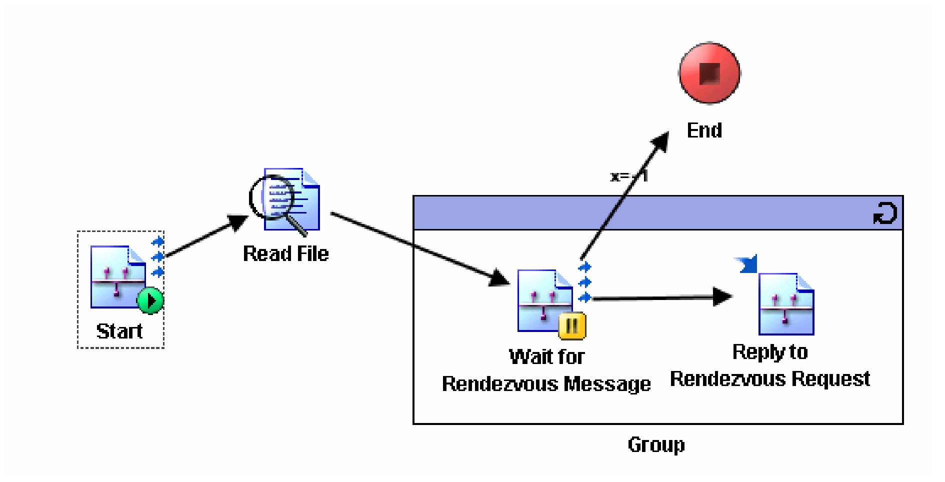


Figure 38 Property Cache

The property cache process illustrated in the figure above reads a property file and maintains the read properties in memory. The other BW jobs can access these properties at runtime by sending queries through Rendezvous messaging. The properties can be changed at runtime by editing the property file.

Applicability

This pattern is applicable when multiple running jobs need to share information with each other.

Related Patterns

Process Synchronization.

5.8 Global Error Handler

Context

This pattern is used in the context of implementing integrated business process using BW Designer interface.

Problem

In an integrated business process, error can happen anywhere during the process flow. Exception must be caught and handling routines must be implemented.

Solution

An exception handling framework implemented using BW can be leveraged by the user processes for exception handling needs.

Please refer to the *TIBCO BusinessWorks Exception Handling Framework* for more details.

Applicability

Always use this pattern as the “standard coding practice” when implementing business processes using BW Designer.

Related Patterns

None.

5.9 Human Approval

Context

Some business processes require human involvement, for instance, to enter a value or to approve something.

Problem

Integration brokers such as BW cannot track human tasks in a process flow directly. Human tasks can be managed by workflow manager instead.

Solution

To communicate with human interface, the business process can send an asynchronous request to human by publishing a RV/JMS message to an application that can subscribe messages in real-time, e.g., a portal application. Then the process waits for response to come back from the human. This is illustrated in the following simple process flow.

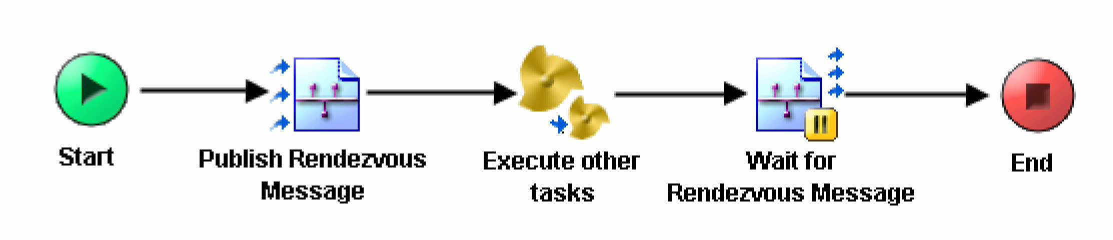


Figure 39Interacting with Human Activity Asynchronously

This implementation is similar to using BW manual activities, but cannot track completion status.

Applicability

Use this pattern to implement business processes that need human intervention.

Related Patterns

Request-reply, Workflow manager.

6 Performance Enhancement Patterns

Performance enhancement patterns include design patterns that are used to improve integration application performance.

In traditional synchronous RPC based world, things are inherently not scalable. Thus the biggest performance enhancement could be made by rewriting synchronous applications into asynchronous ones.

A second consideration to improve application performance is, by no surprise, using multiple threads. Multi-threading is supported by most modern computing platforms and programming languages. For instance, TIBCO's BW and RV all support multiple threading applications.

6.1 Load Balancer

Context

A service provider must be able to serve multiple clients concurrently and scale well as the number of client requests increases per unit of time. This is especially true in many applications that must server a large number of users, such as portals, e-commerce web sites, and etc.

Problem

Scalability is a key issue in building services/servers in that they must be able to scale on multiple concurrent requests.

Solution

Load balancing is usually achieved by sharing load among multiple workers. The messaging middleware supports this feature natively. The messaging middleware dispatches multiple requests to multiple workers based on a configurable load-balancing algorithm or a simple round robin.

RV Distributed Queue (RVDQ) supports load balancing at the transport level using a proprietary algorithm to distribute load (see the following figure). TIBCO EMS uses non-exclusive queue to share queue messages among multiple queue receivers in a round-robin fashion.

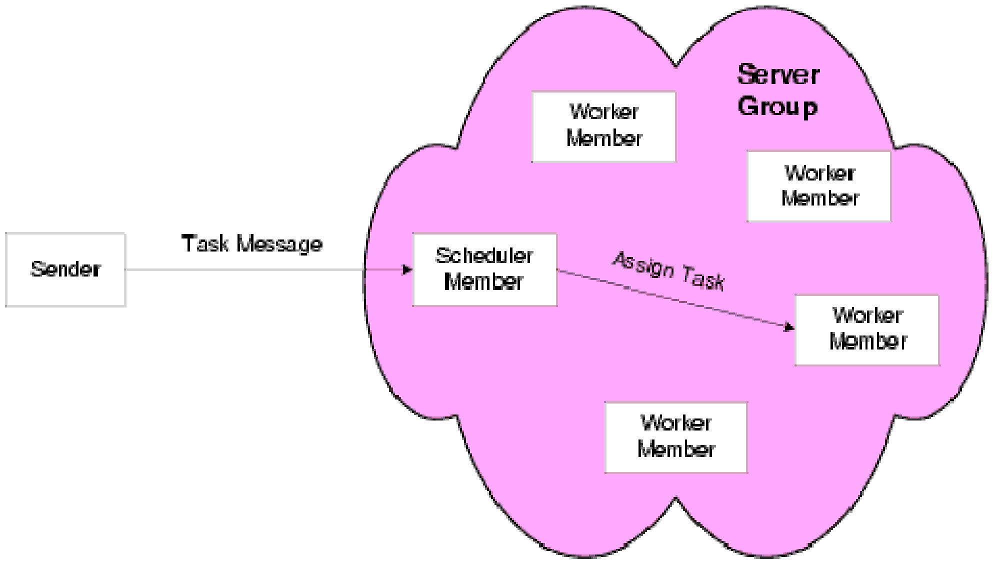


Figure 40 One-of-N Delivery to RV Distributed Queue

An important design consideration when using the Load Balancer pattern is that each member that shares the load must be stateless. In other words, each member that is distributed the work cannot keep the state information of the messages that it processes. For example, if each member must process the message and insert it into a central database, there is no guarantee on the order of the messages that will be inserted into the database from different members. As a result, do NOT use this pattern when state information must be known for each incoming message.

RVDQ versus Multi-threading

Because the RVDQ algorithm requires each member (both worker and scheduler) to keep tracks of each other by exchanging internal status messages, it is not beneficial to have more than 8 members (empirical number) in the queue. This number may be higher on more powerful hardware. At certain number, the internal overhead will overweigh the benefit gained from using this mechanism. There is no such issue for using JMS queues for load balancing.

Alternatively, one can use multi-threading at the message subscriber side to distribute the load using RV or SDK API (by creating multiple event dispatching threads). This approach is more flexible than RVDQ, thus recommended in new application development.

Conclusion

Refer to TIBCO RV Concepts documentation for further details on RVDQ and TIBCO EMS User's Manual for further details on the non-exclusive JMS queue support. Most TIBCO tools such as BW and AE adapters support load balancing by using the RVDQ or JMS non-exclusive queue transport. One can also write a custom application that supports load balancing using one of the messaging transports.

Applicability

Use the Load Balancer pattern to increase the throughput of the server if the server can subscribe to messages. More server instances can be added to the subscriber queue to increase the throughput.

Related Patterns

Publish-subscribe

6.2 Cache

Context

In computer system, cache is an area in memory where a copy of the most up-to-date persisted data is stored for fast access. For example, web servers and portal applications usually cache a lot of information for faster user responses. Similarly, servers can cache information for fast client access as a rule of thumb in general.

Problem

When clients request information from the server, the server will load the information from the database if the server is stateless. However, if the same database record will be accessed multiple times, it is more economical to cache the record in memory for fast client access, avoiding retrieving same data from the database multiple times. For example, a client requests customer name in the first query, and customer address in a subsequent query. Although the customer name and address both locate in the same customer record, the server has to access the database twice to retrieve information.

In addition, in event-driven data publishing, data changes in databases cannot be pushed to the presentation layer in real time efficiently for users to make the right business decisions.

Solution

The solution is to cache information that the server retrieves from the databases for faster access in the future. In demand-driven fashion, the server can retrieve a block of database records into memory at once, avoiding multiple database I/O. Various well-known cache algorithms exist to do this.

In event-driven computing, it is more often that the server subscribes to database updates, i.e., data are pushed from the database to the server in real-time as they change. The Database Adapter system pattern is commonly used to push information as they become available (refer to *Data Access Reference Design* for details). The Transformer system pattern is used to transform the subscribed information to the appropriate format/schema before presenting to the cache manager (this is illustrated in the following figure). The cache manager does the bookkeeping and presents the information to clients if available.

In the following figure, the server will only try to request from the database unless the information is not available in the cache (a “miss”).

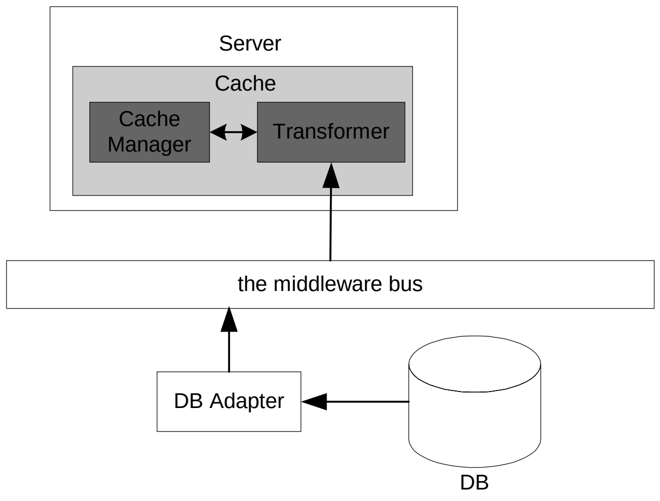


Figure 41 Cache

Depends on the application's tolerance of latency, it is not necessary to always update the cache whenever changes occurs in the DB. The cache can support other kinds of update policies, such as at regular time interval, or based on the workload conditions, etc.

The most often used cache is read-only cache. This makes the cache design fairly simple. Updates always go to the database directly, and cache will be updated accordingly.

Cache can be implemented in BW.

Applicability

Use Cache when same data can to be accessed multiple times, and accessing data from the source is expensive. The application that caches information needs to have enough memory to cache as much information as possible.

Related Patterns

Publish-subscribe, Transformer, Database Adapter

7 System Management Patterns

System management patterns are design patterns used to manage or facilitate the management of integration applications at runtime.

‘TIBCO Hawk Accelerator’ is another service module offered by TIBCO PSG, and consists of more comprehensive Hawk based application monitoring and management support.

7.1 Message Based Monitoring

Context

In a distributed computing environment, applications can be deployed in geographically different locations (e.g., different network, different access restrictions, and etc.). Because distributed applications must work together to deliver business critical information, it is essential to know the health and status of each distributed component to ensure high availability and reliability.

Problem

Collecting information and monitoring status of distributed applications are very difficult because of the distributed nature of the deployment. Centralized server/console based monitoring mechanism use network pull to collect information. Not only is this approach expensive, but also not accurate because it is not real time.

Solution

TIBCO Hawk enterprise monitor is a tool for monitoring and managing distributed applications and operating systems. The software is designed specifically for monitoring distributed systems, so there is no centralized console or frequent polling across the network. With this structure, Hawk is able to scale to multi-thousand node global networks without the use of hierarchical managers and has the flexibility to allow individual managed entities to be added or modified without the need to re-configure or re-start any other parts of the system.

Monitoring information is collected by Hawk *microagents* using the messaging bus. The microagents can be embedded into applications or run as standalone processes (such as Hawk agent and Hawk HMA processes). Hawk Display is the graphical display that subscribes to all Hawk messages from the messaging bus and displays them on the console. The following figure illustrates this design:

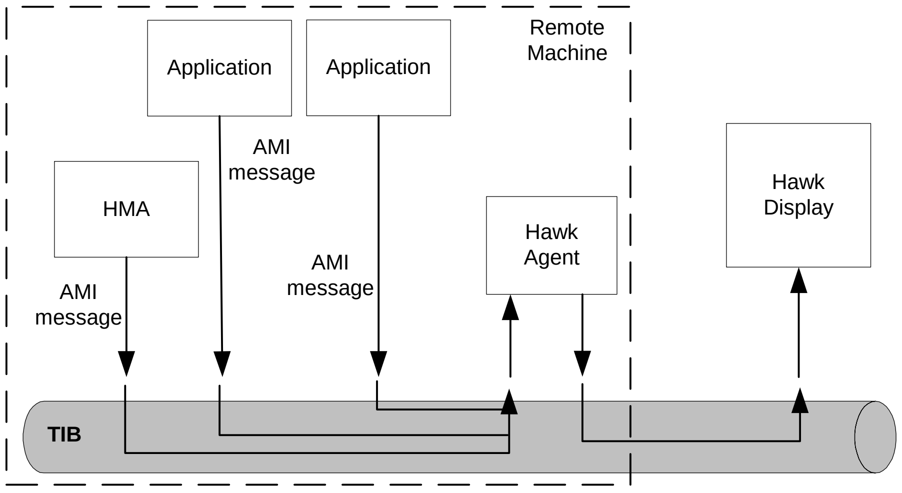


Figure 42 Distributed Monitoring Via TIBCO Hawk

In the above figure, applications (including the Hawk HMA process) are running on a remote machine, each of which is publishing AMI messages to the TIB. Running on each machine is a Hawk Agent, which is receiving AMI messages from the applications and applying a rule base against the messages to generate information of interest to the Hawk display. The applications and the Hawk Agent are also publishing heartbeat messages. The result is that a display of machine-by-machine status can be produced, as well as a drill down into what applications are running on the system, and (if the applications are sending AMI messages) a further drill-down into the state of the applications.

In addition, the reverse path of that indicated in the above figure is possible – via the Hawk Display, the administrator can carry out query commands, and send commands to AMI-enabled applications. There must be a Hawk agent running on each system.

There may also be a Hawk Event Service running on the network, which records all alerts raised and cleared by TIBCO Hawk agents across the network, as well as changes in an agent's alert level.

The messages used by Hawk begin with the subject prefix of "_HAWK", so if the logical diagram above is partitioned into different subnets, messages with that subject must be allowed to pass through. The interaction between the applications and the local Hawk Agent use the facility in RV for "local" messages (indicated by subjects starting with "_LOCAL") so that they will not leave the specific machine.

The main challenge is designing the logic for the Hawk rulebase to determine what conditions for monitor for and what actions to take. Once designed, the rule bases are easy to create, using the GUI of the Hawk display.

Finally, because Hawk uses RV as its underlying infrastructure to deliver alerts, status, there is no restriction on what it can monitor as long as the machine and the network are RV enabled.

Applicability

We recommend using the message-based monitoring system – Hawk to monitor all TIBCO applications (including the infrastructure application).

Related Patterns

Application Instrumentation, Self Recovery

7.2 Application Instrumentation

Context

It is highly beneficial to operational staff to gather as much information as possible from mission critical applications running in production environment.

Problem

When mission-critical applications go down in production, it is difficult to pin down the exact reasons because of the lack of information.

Solution

Application developers can use the TIBCO Hawk Application Management Interface (AMI) API to instrument the applications and send information to Hawk agents periodically. The Hawk AMI protocol is a specification of TIBCO Rendezvous messages that define the interactions between an AMI manager and an AMI application. An AMI manager is any application that monitors or manages AMI applications, or both, via the AMI protocol. A TIBCO Hawk agent is an example of an AMI manager. An AMI application exposes internal methods that allow AMI managers to manage and monitor that application by exchanging data and events with it.

The following list includes some examples of application instrumentation:

- Message queue size
- Track message location in a multi-step process (such as order tracking)
- System resource consumption such as JVM memory usage
- RV ledger content

Applicability

We encourage application developers to use Hawk AMI API to build embedded Hawk microagents in all TIBCO applications.

Related Patterns

Message-based Monitoring

7.3 Integrating Enterprise Management Platforms

Context

In addition to the business process data from TIBCO Hawk, enterprises may also have other infrastructure data obtained through enterprise management (EM) systems such as HP OpenView.

Problem

The business activities are supported by a complex infrastructure of networks, servers, applications, storage devices and security tools. In order to ensure that business performance is optimized, companies must gain insight into the relationship between business activities and the infrastructure that supports them. Therefore, there is a need to integrate the infrastructure and business process data and provide a unified view of the enterprise.

Solution

TIBCO Enterprise Management Advisor enables the integration of business process data (obtained through TIBCO Hawk) with infrastructure data contained in enterprise management platforms such as HP OpenView.

This integration facilitates a bidirectional data exchange between the interfaces of TIBCO and enterprise management platforms. By removing barriers between EM applications and the business integration software that manages business activities, Enterprise Management Advisor makes it possible for business managers and IT staff to analyze developing situations, identify and initiate the best course of action, and take steps to ensure business continuity and the satisfaction of service level agreements.

Applicability

We encourage the use of *TIBCO Enterprise Management Advisor* if enterprise management systems such as HP OpenView are in use. This would provide a unified view of the enterprise and the ability to correlate business process data with the infrastructure data.

Related Patterns

Message-based Monitoring

7.4 Self Recovery

Context

Runtime application monitoring and operations management.

Problem

In production, business critical applications are required to run 24x7. When the application goes down unexpected (process failure or hardware crash), manual recovery is usually inefficient (slow) and costly. We must be able to automatically detect when a crucial app is down and re-launch it as soon as possible based on pre-defined logic.

Solution

Use TIBCO Hawk to build the following self recovery rules to self-recover from failed runtime processes:

- Use *GetInstanceCountByCommand* method provided by the default Hawk rulebase to know how many instances of the app run in the studied system. If it is "0" launch a script from Hawk to re-launch it.
- Build a script that detects whether an app is running and a second script to re-launch the app. Call the first script to return a string or a number and the second if the result doesn't match your expectations.

Normally when we detect a production instance down, we send an email to notify appropriate person; when the instance is re-started (successfully or with failure), we send another email notification. The email notification can go through an email server, say, Microsoft Exchange, to delegate to the appropriate roles.

This technique can be used to recover runtime processes at a different level than those managed by cluster software or hardware.

Another example of self recovery is to allocate more system resource automatically when the current consumption on a particular system resource (such as a message queue, or memory) is low. This is very similar to the ATO (automatic throughput optimization).

Applicability

Using Hawk based self-recovery patterns can build very powerful rules to recover from exception conditions that need no human intervention.

Related Patterns

Message-based Monitoring

7.5 Centralized Logging

Context

In a distributed computing environment, it is useful to collect information from different applications and analyze the logs collectively or save the logs for reporting or future analysis.

Problem

Logging by individual applications in a distributed environment create many log files. Individual log files proliferate the system, resulting in management headache.

Solution

Publish status or logs from each application to the middleware bus using the *Publish-subscribe* pattern. Use *Database Adapter* pattern to subscribe to the log messages write the logs to the central log database, or the Write File activity in BW to write to a log file when the logging database is not available. The following figure shows this design.

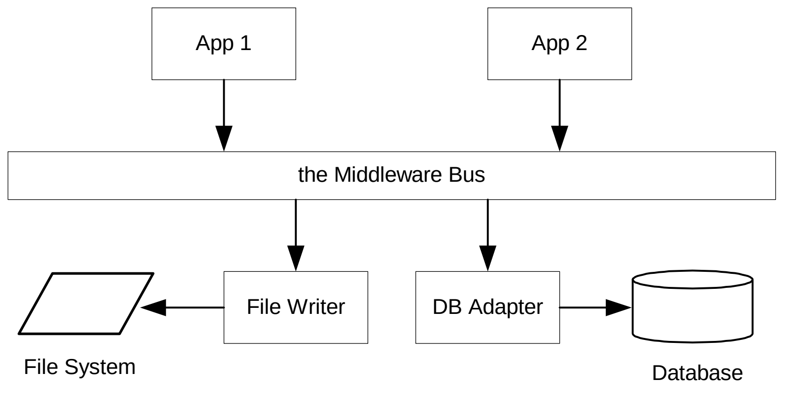


Figure 43 Centralized Logging

Reporting tools can be applied to the resulting file or database based logs to generate reports. (*Log File Scanning* system pattern can be used to scan the central log file). It is important to design a standard log message schema and subject name for all applications. The log message should consist of at least the following information:

- Log level
- Location: application name/module/line
- Error code/description

Applicability

Centrally managed audit logs for reports and analyses.

Related Patterns

Publish-subscribe, Database Adapter, Log File Scanning

7.6 Log File Scanning

Context
Most applications generate log files. Log file contains information about errors, warnings, and information on the application status.

Problem
Write log file scanning utilities is a repetitive effort, especially when you have to write one for each application. Deploying, running, and managing these utilities at different locations is even worse.

Solution
Write a utility to scan keywords (such as “ERROR”) in a log file and build a Hawk rulebase to react based on different keyword findings. The Hawk rules can be easily distributed to all the Hawk managed nodes using the Hawk console.

Applicability
Each product generates a log file. Refer to respective product manuals for log file default location and content.

Related Patterns
Centralized Logging

7.7 Automatic Throughput Optimization

Context

This is used to dynamically adjust throughput in a RV/JMS publish-subscribe scenario.

Problem

In a fast producer, slow consumer RV/JMS publish-subscribe scenario, subscribers cannot keep up with the publisher. One solution is to queue messages that cannot be processed in time by subscribers somewhere, or using a queue-based messaging solution alternatively.

Solution

The automatic throughput optimization is a mechanism to slow down, suspend, or stop the publisher automatically to allow the subscriber to keep up.

The subscriber monitors certain parameter, such as the number of messages accumulated in the incoming RV/JMS message queue, or its memory consumption. When the parameter reaches a threshold level, it sends a signal message to the publisher to let it slow itself down. Once the subscriber catches up, it sends another wake-up message to the publisher to let it resume normal publishing.

Note that the publisher usually cannot be slowed down or suspended if it is publishing to many subscribers. One slow subscriber cannot ask the publisher to slow down. For instance, this happens in manufacturing factory floor applications.

Applicability

Use this in a fast producer, slow consumer RV/JMS publish-subscribe scenario, in which the publisher can be slowed down, suspended, or stopped to let the consumer catch up.

Related Patterns

Publish-subscribe

7.8 Heartbeat Generator

Context
Remote applications, especially those who are not accessible, need to report their status on whether they are still alive.

Problem
It is difficult to use traditional methods to pull status of a running application over the network, especially those with access restrictions. In addition, sometimes it is not easy to tell whether an application is available even if the application process exists.

Solution
Use the Hawk AMI API to implant a Hawk microagent in the application and generate a heartbeat message (that contains a simple string such as “I am alive” and the application name/location) to publish to Hawk. One can then combine the *Centralized Logging* and *Log File Scanning* or use the *Message-based Monitoring* patterns to monitor the application health.

This pattern could be useful for applications that do not have built-in Hawk microagents and can be used on all platforms (as long as they support Java).

The design should specify the heartbeat message format, which would normally consist of timestamp, origin, and etc.

To use this pattern together with Hawk rules, Hawk RV message adapter needs to be used to subscribe to heartbeat messages. Hawk RV message adapter comes with the standard Hawk software as a separate download.

Applicability
When there is a need to know whether the application is alive.

Related Patterns
Publish-subscribe, Message-based Monitoring