Integration Patterns Applied to Print Production

Claes Buckwalter[*]

Keywords: messaging, integration, pattern language, JMF, JDF

Abstract: In a modern printing plant software systems are ubiquitous and indispensable. Systems for order management, production planning, and other administrative systems are implemented in software. Most, if not all, equipment on the plant floor has a software front-end that either controls the physical equipment directly or displays instructions for a human operator to interpret and execute. These software systems are not isolated islands. During production they need to communicate and exchange information. For example, a prepress workflow system may send configuration parameters to production equipment and production equipment may send status updates to production monitoring systems. This type of communication is typically implemented by sending messages, discrete units of data, between the systems.

Integrating heterogeneous systems using messaging is nothing unique to the printing industry. It is a well-proven solution and there are several general-purpose solutions available for integrating disparate systems using messaging. The experience and knowledge on the subject has been documented in several pattern languages.

This paper examines Job Definition Format's messaging protocol Job Messaging Format (JMF), and maps JMF concepts to patterns found in pattern languages for system integration using messaging. Weaknesses found in JMF are discussed and patterns are applied to suggest alternative solutions. The result is JMF expressed using general enterprise integration patterns.

[*] Digital Media division, ITN, Linköping University, Sweden

1 Introduction

Software systems play a vital role in a modern printing plant. Management Information Systems (MIS), workflow systems, and prepress systems that process digital content are implemented entirely in software. Most equipment on the plant floor has a software front-end that controls the physical equipment directly or displays instructions for human operators to interpret and execute.

Efficient print production requires that the software systems can communicate. Digital content, job specifications, configuration parameters, and tracking data need to be exchanged between systems continuously during production. Integrating the heterogeneous systems in the printing plant is crucial.

The integration technology with the widest acceptance in the printing industry is *Job Definition Format (JDF)*, maintained by the industry consortium CIP4 [7]. The JDF specification [4] defines an XML-based data format, a "job ticket format", that allows systems in the print production workflow to describe printed products, and the workflow required to produce them, in a standardized manner. Included in the JDF standard is an XML-based communication protocol called *Job Messaging Format (JMF)*. JMF allows systems to exchange units of data, messages, containing control commands, JDF job ticket data, and tracking information.

Integrating disparate systems using messaging is not unique to the printing industry; the principles behind JMF are similar to those of general-purpose integration solutions such as Web Services and message-oriented middleware [10]. Best practices and solutions to reoccurring problems within the area of system integration using messaging have been documented in several *pattern languages* [1, 3, 8, 10]. Originally used in architecture [2], then in object-oriented software design [9], a pattern language is a collection of reusable solutions, patterns, which can be used to solve problems in a particular problem space.

This paper provides an analysis of JMF and classifies the concepts found in JMF using patterns found in patterns languages for system integration and service interaction [1, 3, 8, 10]. JMF supports the concept of message routing performed by intermediate systems, for example a workflow system that routes messages to/from systems under its controls. This is beyond the scope of this paper, which focuses on direct point-to-point communication between systems without an intermediary.

2 The Communication Models of JMF

The JDF specification [4] defines two communication models for JMF messaging: "unidirectional" and "bidirectional" messaging. Most JMF messaging is of request-reply type where a system sends a request message and gets a reply message back. The bidirectional communication model of JMF uses the HTTP protocol for message transport. HTTP is a bidirectional in the sense that a system sending a request gets an immediate reply on the same communication channel as the request was sent. The unidirectional communication model uses files for message transport. A system sends a request by writing a file to a location and receives a reply by reading a file from another location. The file-based protocol is unidirectional in the sense that a request does not receive an immediate reply. Instead, the reply is received over a separate channel.

This section classifies the two communication models of JMF based on the level of decoupling between sender and receiver.

2.1 *File-Based JMF Messaging*

Exchanging information between systems by reading and writing files is a common integration style. In the graphic arts industry this integration style is called "hot folder" integration and has a long history of use in prepress workflows where it is used to automate the exchange and processing of content files, such as PDF documents.

In a file-based JMF messaging scenario each system is configured with an input folder that it uses to receive files containing JMF messages. A system's input folder can be regarded as the channel that the system uses to receive messages from one or more other systems. A system sends a message by writing a file containing the message to the input folder of the receiving system. The sender's message contains a URL that specifies a file (a channel) to which the receiver can write a reply message.

The JDF specification [4] describes file-based JMF messaging as *unidirectional* and *asynchronous*. Unidirectional in that separate channels are used for sending and receiving messages; a single channel is never use for both. Asynchronous in that a system can send a message without requiring that the receiver be available at the time of sending. The sender and receiver are decoupled by the file system that provides an infrastructure for messaging while ensuring a loose coupling between systems. The file system can be viewed as a primitive form of *message-oriented middleware* or *messaging system* [10].

2.2 *HTTP-Based JMF Messaging*

The *Hypertext Transfer Protocol (HTTP)* is the network transport protocol used to transfer web pages between web servers and web browsers.

HTTP is also used by Web Services standards as the transport protocol for exchanging business data between heterogeneous and distributed applications. JMF has several similarities to Web Services standards.

HTTP-based JMF messaging requires that each system implement a HTTP client and a HTTP server. The sender of a message uses its HTTP client to open a *HTTP connection* to the address of the receiver's HTTP server. A *HTTP request* containing a JMF message is sent over the connection. The receiver's HTTP server receives the request, process the message and replies using a *HTTP response* containing a reply JMF message. The request and reply messages are both sent over the same HTTP connection.

HTTP-based JMF messaging is described by the JDF specification as *bidirectional* and *synchronous*. The communication is bidirectional in that the HTTP connection the sender uses to connect to the receiver is used to send both the request JMF message and the reply JMF message. The communication is synchronous in that both sender and receiver must be available at the time of sending and that the sender blocks until the request message is sent and a reply received back. This type of communication is often called *Remote Procedure Call (RPC)* [10] because it is similar to invoking a procedure or method in a programming language. JMF messaging is not the only example of using the HTTP protocol for RPC communication; most Web Services standards, for example SOAP [11], use this approach.

The synchronous nature of HTTP-based JMF messaging, and RPC in general, entails a tight coupling between the systems involved in the communication.

2.3 *Coupling*

Coupling is an important concept that applies at all levels in software design. It is generally considered good design to strive for as loosely coupled, *decoupled*, software components that have a minimum number of dependencies on each other. Systems assembled of loosely coupled components tend to be flexible and allow a component to be modified or replaced with a minimum of impact on other components. The JDF specification's classification of file-based JMF messaging as "asynchronous" and HTTP-based JMF messaging as "synchronous" indicates the degree of coupling between systems in these two types of communication. However, a more precise classification of the two types of JMF messaging is desirable.

Eugster et al. [8] have identified three dimensions of (de-)coupling in the domain of communication middleware. These dimensions have been formalized by [1] and used to classify common middleware solutions. The three dimensions of decoupling are:

- *Time decoupling* – the sender and receiver of a message do not need to be active at the same time.
- *Space decoupling* – a message is sent to a symbolic address, not the direct address of a system.
- *Synchronization decoupling* – senders do not block while sending a message and receivers are notified by a callback when a new message is available.

Using these three dimensions the two communication models of JMF messaging can be classified, see Table 1. A set of notational elements for each possible combination of couplings has also been developed [1]. The notations corresponding to file-based and HTTP-based JMF messaging are shown in Figure 1.

Table 1 Decoupling of file-based and HTTP-based JMF messaging

	Time decoupling	**Space decoupling**	**Synchronization decoupling**
File-based JMF	Yes	Yes	Blocking send Blocking receive
HTTP-based JMF	No	No	Blocking send Non-blocking receive

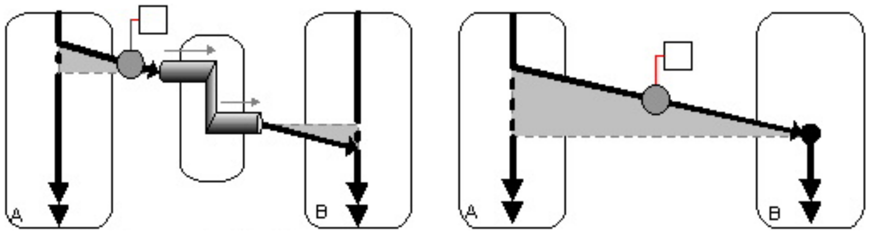

Figure 1 Decoupling configuration notations [1] for file-based (left) and HTTP-based (right) JMF messaging

File-based JMF messaging provides the highest degree of decoupling between communicating systems and has a classification similar to message-oriented middleware systems [1]. The file system acts as a primitive messaging system [10] that decouples sender and receiver. A sender can write a message file to the file system without the receiver being active. A sender sends a message by writing a file to a URL which points to a path in a file system, not directly to the receiving system. When a system sends a JMF message the thread sending the message typically blocks until the file has finished being written to the file system. To receive messages a system typically has a thread that scans the file system at a specified time interval – the *Polling Consumer* pattern [10].

When the thread discovers that a file it blocks until the file is read, the message processed, and possibly a reply message sent. To summarize, file-based JMF provides space decoupling and time decoupling but the blocking send and receive characteristics result in limited synchronization decoupling.

Systems participating in HTTP-based JMF messaging are both space coupled and time coupled. A system sending a JMF message connects directly to the receiver and both systems are active during the delivery of the message. The message-sending thread of the sender typically blocks until the message has been sent and a reply is received. Receiving a JMF message is typically non-blocking. For each message received by a system a new thread is started that handles message processing and the sending of a reply – the *Event-Driven Consumer* [10] pattern.

While HTTP-based JMF messaging inflicts a higher coupling between communicating systems it is often considered less complex to implement the messaging endpoints of a system with these characteristics. The blocking send combined with time coupling relieves the sender of managing any state information related to the message being sent. This is managed by the programming language's call stack. In a time decoupled system the sender would have to store the context to which the message is related so that when a reply is later received the context can be recreated and processing continue.

In practice, HTTP-based JMF messaging is the communication model implemented by the majority of JDF-enabled systems. The JDF specification [4] implicitly favors HTTP messaging and the Base ICS [5] explicitly requires that systems of base conformance level 2 and upwards support HTTP-based messaging. In fact, file-based JMF messaging is not required at all by the Base ICS. Instead, a hot folder exchange of JDF instance files is required that is based on the same principles as file-based JMF messaging.

The rest of this paper describes JMF from the point of view of HTTP-based JMF messaging.

3 Patterns in JMF

Two pattern languages that describe the integration and interactions of distributed heterogeneous systems are [10] and [3]. This section uses these two pattern languages to examine JMF and identify where the design choices made for JMF correspond to the documented patterns. The patterns identified are summarized in Appendix A.

3.1 Message Format

In essence, the JDF job ticket format and Job Messaging Format (JMF) are *Canonical Data Models* [10] that standardize the data exchange between systems in the print production workflow. Each system may use its proprietary data model internally, but must be able to export and import data according to the canonical data model defined by the JDF specification [4].

As JDF evolves, new versions of the specification are released. Each new version of the JDF specification results in a new version of the XML schema that defines the syntax of the JMF message format. In addition, a JMF message can be conformant with several *Interoperability Conformance Specifications (ICS)* [6], each of which may have multiple versions. For a system to know how to process a message it needs to able to identify the version and format of the message received. JMF solves this by implementing the *Format Indicator* [10] pattern. Each JMF message contains version number fields and identifies the XML schema(s) that the message's syntax adheres to.

The fundamental JMF messaging interaction is point-to-point request-reply and can be mapped to the *Send/Receive* [3] and *Remote Procedure Invocation* [10] patterns. A request message is sent by one system and received by another system; the receiver processes the message and sends a reply message back to the requestor. In a HTTP-based JMF messaging scenario the reply message can be sent synchronously on the same channel as the request message or asynchronously on a separate channel.

Implementing the *Send/Receive* pattern requires the each message be uniquely identifiable. All JMF messages therefore have a unique identifier, a message ID. A system receiving a reply to a request message it previously sent must be able to correlate the reply message with the original request. Therefore, a system sending a reply is required to specify the ID of the original request message in the reply message. This corresponds to the *Correlation Identifier* [10] pattern.

In the case of a synchronous message reply, the replier sends the reply message over the same channel as it received the request message; the reply message is sent using a *HTTP response* to the request message's *HTTP request* over the same *HTTP connection*. If the requesting system accepts asynchronous replies, the request message contains a *Return Address* [10] that may be used by the receiving system to send the reply message; in this case the request and reply are sent over separate channels.

Some JMF message types are used to move large amounts of data between systems. An example is the JMF *SubmitQueueEntry* message that is used to submit a print job, consisting of a JDF instance file and high-resolution content files, to a system for execution. JMF supports sending all files related to the print job in a single large message called a *MIME package* [4]. However, this is often an inefficient approach, especially in the case where the print job, including all related files, must be delegated to another system for execution. As a more efficient alternative, JMF implements the *Claim Check* [10] pattern. Instead of sending the JDF instance file and the content files in a single JMF message, a system can choose to send a JMF message containing a URL, a *claim check*, which points to the JDF instance file. The JDF instance file, in turn, contains URL references to the content files. The receiving system can use the claim check URL to first retrieve the JDF instance file, and then retrieve the relevant content files.

3.2 *JMF Message Families*

The JDF specification [4] divides JMF messages into six message families, each family consisting of messages of different types used for specific purposes. For example, the message of type *QueueStatus* falls under the *Query* message family and is used to request the state of a system's queue. There are 44 message types in total.

The six message families of JMF correspond to three message construction patterns documented in [10]. The *Command Message* pattern is used to invoke a procedure in another system. The *Document Message* pattern is used to transfer data between systems. The *Event Message* pattern is used to asynchronously notify systems of events.

This section identifies the patterns found in the six message families of JMF.

3.2.1 JMF Query

A JMF *Query* message requests information about the state of a system. The caller may specify a *Return Address* [10] allowing the callee to process the *Query* request asynchronously. The caller may also specify a subscription for the *Query* requiring the callee to send a *Signal* message on a certain time interval or when a specific state is reached.

A JMF *Query* message is the *Command Message* [10] pattern.

3.2.2 JMF Command

A JMF *Command* message requests a change in a system's state. The caller may specify a *Return Address* [10] allowing the callee to process the *Command* request asynchronously.

A JMF *Command* message is the *Command Message* [10] pattern.

3.2.3 JMF Response

A JMF *Response* message is a mandatory synchronous reply to a *Query* or *Command* request notifying the caller that the request has been received. A *Response* may or may not contain the results of the processed *Query* or *Command* request.

A JMF *Response* message is the *Document Message* [10] pattern.

3.2.4 JMF Acknowledge

A JMF *Acknowledge* message is an optional asynchronous reply to a *Query* or *Command* request. If the caller of the request specified a *Return Address* [10] the callee may choose to process the request asynchronously and send one to three *Acknowledge* messages notifying the caller of the completed stages of message processing (*Received*, *Applied*, and *Completed*). The final *Acknowledge* message sent by the callee must contain the results of the processed *Query* or *Command*.

A JMF *Acknowledge* message is a combination of the *Document Message* and *Event Message* patterns [10].

3.2.5 JMF Signal

A JMF *Signal* message is an asynchronous reply to a *Query* message sent on a certain time interval or when a specific state is reached. *Signal* messages are sent to the *Return Address* [10] specified in the *Query* request that initiated the subscription.

A JMF *Signal* message is the *Event Message* [10] pattern.

3.2.6 JMF Registration

A JMF *Registration* message requests that the callee send JMF *Command* messages to a third party on a certain time interval or when a specific state in the callee system is reached.

A JMF *Registration* message is the *Command Message* [10] pattern.

3.3 *JMF Message Interactions*

The interactions of JDF-enabled systems using the six message families of JMF map to five of the interaction patterns defined in [3].

A system sends a *Query* or *Command* message to a system and receives a *Response* message as a reply. This corresponds to the *Send/receive* [3] and *Remote Procedure Invocation* [10] patterns, illustrated in Figure 2.

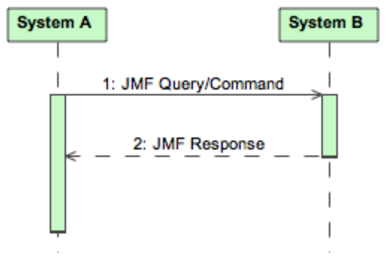

Figure 2 Send/Receive pattern in JMF

The extension of the above interaction that includes asynchronous *Acknowledge* messages is a variant of the *Multi-responses* [3] pattern, Figure 3.

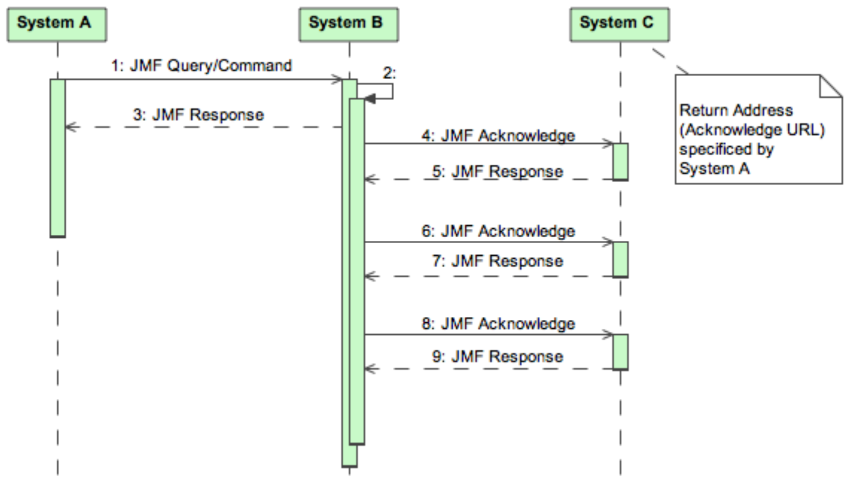

Figure 3 Multi-responses pattern with Acknowledge

Another JMF interaction that fits the *Multi-responses* [3] pattern definition is a *Query* message specifying a subscription; see Figure 4.

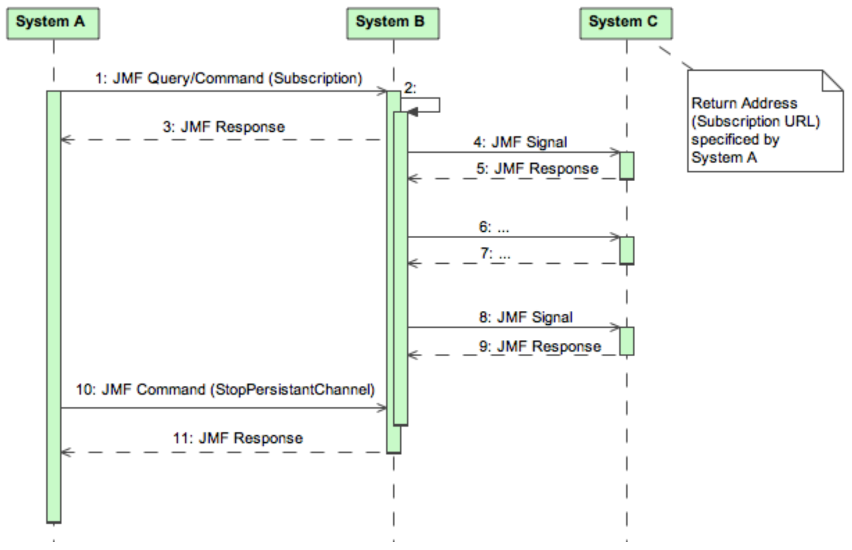
System A
System B
System C
1: JMF Query/Command (Subscription)
2:
3: JMF Response
4: JMF Signal
5: JMF Response
6: ...
7: ...
8: JMF Signal
9: JMF Response
10: JMF Command (StopPersistantChannel)
11: JMF Response
Return Address (Subscription URL) specificed by System A

Figure 4 Multi-responses with Signal

The sending of a *Signal* or *Acknowledge* message corresponds to the *Send/Receive* [3] pattern. The reception of a *Signal* or *Acknowledge* corresponds to the *Receive/Send* [3] pattern. To be more specific, the *Send* part of this incarnation of the *Receive/Send* pattern only requires that the receiver send a JMF message if it encounters a message-processing fault. If the receiver processes the JMF message successfully, it is not required to send a JMF message as a reply. However, in the case of HTTP-based JMF messaging a HTTP response with HTTP status code 200 (the request has succeeded) must be sent.

The sending of a *Registration* and the resulting interactions map to the *Request with referral* [3] pattern, see Figure 5. Any JMF message that uses a *Return Address* [10] can be considered to map to the *Request with referral* pattern. For example, the return address of a *Query* subscription could refer to a third party instead of the sender of the Query containing the subscription.

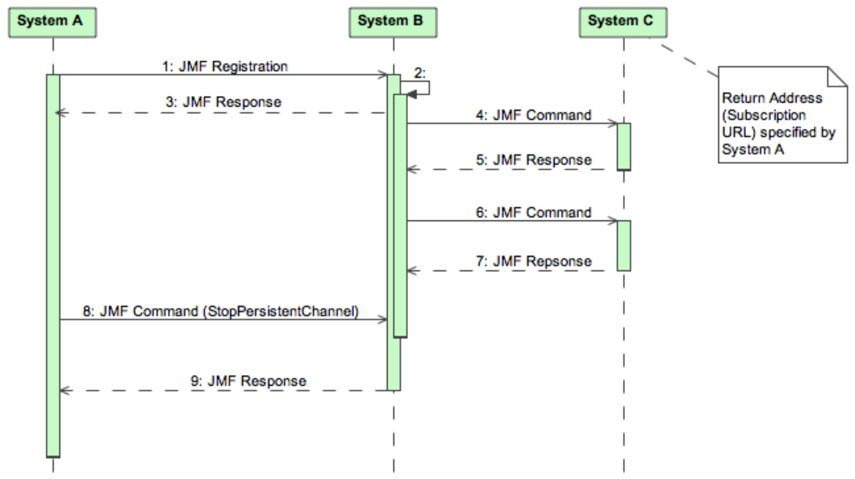

Figure 5 Request with referral in JMF

4 Conclusions and Discussion

File-based JMF messaging and HTTP-based JMF messaging both have weaknesses. The space and time decoupling of systems provided by file-based JMF messaging is a favorable property it shares with message-oriented middleware solutions [1, 10]. However, while message-oriented middleware systems are designed specifically for messaging, file-based JMF messaging uses conventional file systems not designed with the intention of serving as a messaging infrastructure.

HTTP-based JMF messaging lacks both space and time decoupling: the sender connects directly to the receiver, requiring that both systems be active during the interaction. Nonetheless, the simplicity of HTTP's request-reply point-to-point communication is what has made it into the ubiquitous protocol it is and the protocol used by both Web Services [10] standards and JMF to integrate distributed systems.

In practice, HTTP-based JMF messaging is the communication model that JDF-enabled systems implement. The JDF specification [4] implicitly favors HTTP and the ICSs [6] explicitly require that systems implement JMF messaging over HTTP in order to be conformant.

Although HTTP-based JMF messaging enforces a tight coupling between the message endpoints of interacting systems, there is nothing preventing a system from internally implementing a loose coupling to its endpoint. By adding an intermediate layer between a system and its message endpoint the system can be decoupled from the messaging technology, see Figure 6.

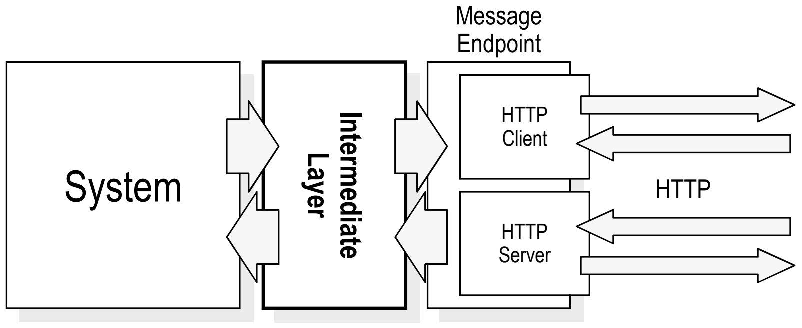

Figure 6 Decoupling of system and message endpoint

When sending a message a system would call the intermediate layer. The call would return immediately and it would be left to the intermediate layer to take responsibility for the delivery of the message to the receiver. Likewise, when receiving a message a system's message endpoint would call the intermediate layer, which would determine further processing of the message. The intermediate layer could be designed to provide full decoupling between the system and its message endpoint, see Table 2.

Table 2 Decoupling properties

	Time decoupling	**Space decoupling**	**Synchronization decoupling**
Intermediate Layer	*Yes*	*Yes*	*Non-blocking send* *Non-blocking receive*
File-based JMF	Yes	Yes	Blocking send Blocking receive
HTTP-based JMF	No	No	Blocking send Non-blocking receive

An intermediate layer could also support the notion of *reliable delivery* of JMF messages, a topic covered by neither the JDF specification nor the ICSs. Reliable delivery is concerned with guaranteeing the delivery of messages, in the order they were sent, and without duplicates. The JDF specification defines the necessary constructs to support guaranteed delivery and delivery without duplicates. The *Multi-responses* [3] pattern with *Acknowledge* messages (Figure 3) can be used to implement guaranteed delivery of messages, and a receiver can eliminate message duplicates by keeping a log of the IDs of received messages. However, additional constructs need to be added to the JDF specification to support ordered delivery of messages.

5 Acknowledgments

The author wishes to thank professor Björn Kruse, Linköping University, and the members of the CIP4 organization for their feedback and support.

6 References

1. L. Aldred, W. M.P. van der Aalst, M. Dumas, and A.H.M. ter Hofstede. *On the Notion of Coupling in Communication Middleware*, Lecture Notes in Computer Science, Volume 3761, pp. 1015 – 1033. Springer-Verlag Berlin Heidelberg, 2005.
2. C. Alexander, S. Ishikawa, M Silverstein. *A Pattern Language* - *Towns Buildings Construction*. Oxford University Press, 1977.
3. A. Barros, M. Dumas, A.H.M. ter Hofstede. *Service Interaction Patterns: Towards a Reference Framework for Service-based Business Process Interconnection*. Technical Report FIT-TR-2005-02, Faculty of Information Technology, Queensland University of Technology, Brisbane, Australia, March 2005. http://sky.fit.qut.edu.au/~dumas/ServiceInteractionPatterns.pdf, accessed March 2006.
4. CIP4. *JDF Specification Release 1.3*, 2005. http://www.cip4.org/documents/jdf_specifications/JDF1.3.pdf, accessed March 2006.
5. CIP4. *Base Interoperability Conformance Specification (Base ICS) 1.0 rev A*, 2005, http://www.cip4.org/document_archive/documents/ICS-Base-1.0RevA.pdf, accessed March 2006.
6. CIP4. *Interoperability Conformance Specification (ICS) Registry*. http://www.cip4.org/document_archive/ics.php, accessed March 2006.
7. CIP4. *The International Cooperation for the Integration of Processes in Prepress, Press and Postpress Organization (CIP4) home page.* http://www.cip4.org, accessed March 2006.
8. P.Th. Eugster, P.A. Felber, R. Guerraoui, and A.-M. Kermarrec. *The Many Faces of Publish/Subscribe*. ACM Computing Surveys, 35(2), pp. 114/131. June 2003.
9. E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
10. G. Hophe and B. Woolf. *Enterprise Integration Patterns*. Addison-Wesley, 2004.
11. *Simple Object Access Protocol (SOAP) Version 1.2*, 2003. http://www.w3.org/TR/soap12-part1/, accessed March 2006.

Appendix A

Pattern Name	Pattern Occurrence in JMF
Canonical Data Model	Job Messaging Format (JMF) Job Definition Format (JDF)
Command Message	JMF Command JMF Query
Document Message	JMF Response JMF Acknowledge
Event Message	JMF Signal
Format Indicator	*JMF/@Version* *JMF/@ICSVersions* *JMF/@xmlns* *JMF/*/@xsi:type* HTTP header *Content-type*
Correlation Identifier	*JMF/*/@ID* *JMF/*/@refID*
Return Address	*JMF/Query/@AcknowledgeURL* *JMF/Command/@AcknowledgeURL* *JMF/Query/Subscription/@URL* *JMF/Registration/Subscription/@URL* *JMF/Command[@Type='SubmitQueueEntry'] /QueueSubmissionParams /@ReturnJMF* *JMF/Command[@Type='SubmitQueueEntry'] /QueueSubmissionParams /@ReturnURL* *JDF/NodeInfo/@TargetRoute*
Claim Check	*JMF/Command[@Type='SubmitQueueEntry'] /QueueSubmissionParams /@URL* *JMF/Command[@Type='ReturnQueueEntry'] /ReturnQueueEntryParams /@URL* *JMF/Command[@Type='ResubmitQueueEntry'] /ResubmissionParams/@URL*