XML Protocol Abstract Model

W3C Working Draft 9 July 2001

This version:
http://www.w3.org/TR/2001/WD-xmlp-am-20010709/
Latest version:
http://www.w3.org/TR/xmlp-am/

Editors:
Stuart Williams (Hewlett-Packard Company)
Mark Jones (AT&T Labs)
Contributors: (alphabetical)
Mark Baker (Sun Microsystems)
Martin Gudgin (DevelopMentor)
Oisín Hurley (Iona)
Marc Hadley (Sun Microsystems)
John Ibbotson (IBM Corporation)
Scott Isaacson (Novell Inc.)
Yves Lafon (W3C)
Jean-Jacques Moreau (Canon)
Henrik Frystk Nielsen (Microsoft Corporation)
Krishna Sankar (Cisco Systems)
Nick Smilonich (Unisys)
Lynne Thompson (Unisys)

Abstract

This document describes an Abstract Model of XML Protocol.

The challenge of crafting a protocol specification is to create a description of behaviour that is not tied to any particular approach to implementation. There is a need to abstract away from some of the messy implementation details of buffer management, data representation and specific APIs. However, in order to describe the behaviour of a protocol one has to establish a set of (useful) concepts that can be used in that description. An abstract model is one way to establish a consistent set of concepts. An abstract model is a tool for the description of complex behaviour – it is not a template for an implementation... although it should not stray so far away from reality that it is impossible to recognise how the required behaviours would be implemented.

Status of this Document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. The latest status of this document series is maintained at the W3C.

This is the first W3C Working Draft of the XML Protocol Abstract Model for review by W3C members and other interested parties. It has been produced by the XML Protocol Working Group (WG), which is part of the XML Protocol Activity.

The XML Protocol Working Group has developed the Abstract Model in order to provide a useful framework for the evaluation of candidate protocols and for reasoning about the development of the protocol itself.

At this time the XML Protocol Working Group has not decided whether an Abstract Model such as this one will be eventually published as a separate Note, a separate Recommendation or whether material from the Abstract Model will be incorporated as non-normative (informative) text within an eventual Recommendation specifying an XML Protocol. The Working Group solicits feedback on the question of whether or not to include a model such as this in an eventual Recommendation.

This document currently uses the term "XML Protocol", or the short form "XMLP", to refer to the protocol being modelled.

The XML Protocol Working Group maintains an issues list [Issues] that contains descriptions of concerns raised against the Abstract Model. As part of its continuing work, the XML Protocol Working Group will resolve outstanding issues that concern the reconciliation of differences between the Abstract Model and the SOAP version 1.2 specification.

Comments on this document should be sent to the W3C mailing list xmlp-comments@w3.org (public archives).

Discussion of this document takes place on the public <xml-dist-app@w3.org> mailing list (Archives) per the email communication rules in the XML Protocol Working Group Charter.

This is a public W3C Working Draft for review by W3C members and other interested parties. It is a draft document and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use W3C Working Drafts as reference material or to cite them as other than "work in progress". A list of all W3C technical reports can be found at http://www.w3.org/TR/.

Table of Contents

1. Introduction

An abstract model is a useful means to develop a description of a system. It abstracts away from practical details such as specific API definitions, data representation, and buffer management. It provides a way to give a precise description of the externally visible behaviour without being prescriptive of implementation architecture.

This document is intended to serve as an overview and introduction to the XML Protocol and its framework.

Section 2 presents an overview of the abstract model

Section 3 presents a model for the services provided by the XML protocol layer to XML protocol applications.

Section 4 presents a model for the extensible processing of XML protocol messages.

Section 5 presents a model for the binding of XML protocol to underlying protocol layers.

1.1 Definition of Terms

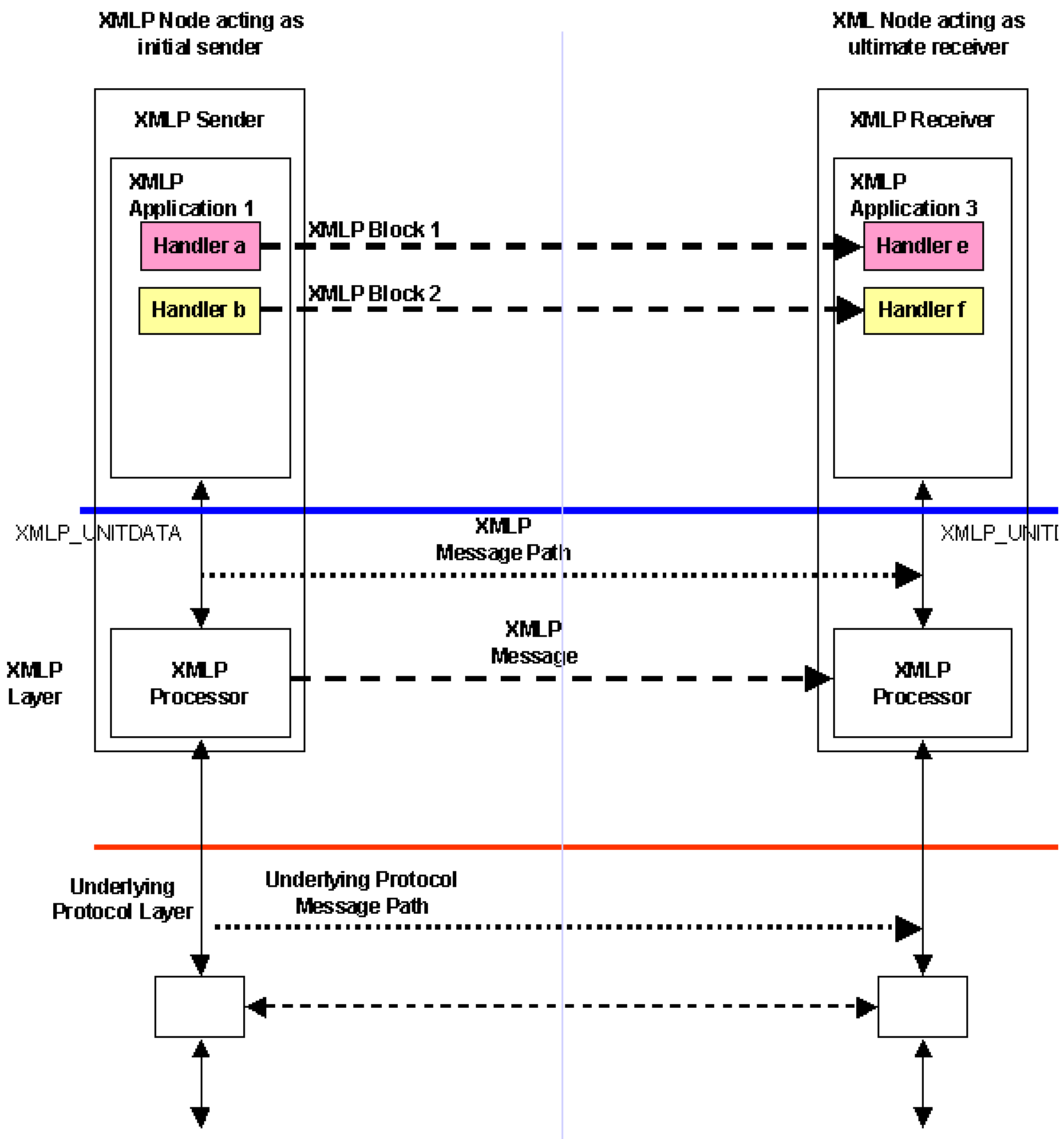
This document makes use of terms defined in the [XMLPReqs]. Additional terms introduced in this document are defined locally in this section, however, in the long term we anticipate that they will be incorporated into a single glossary for all documents produced by the WG.

XMLP Application	A client or user of the services provided by the XML Protocol Layer. An XML Protocol Application may act in the initiating or responding role with respect to two-way request response operations and in the sending or receiving roles with respect to one-way operations. XML Protocol Applications may also act in an intermediary role with respect to both two-way and one-way operations. XML Protocol Handlers are encapsulated within XML Protocol Applications.
XMLP Layer	The XML Protocol Layer is an abstraction that provides services or operations that transfers packages of XML Protocol Blocks between peer XML Protocol Applications via zero or more XML Protocol Intermediaries.

XMLP Operation	A primitive capability or service offered by the XML Protocol Layer. The XML Protocol Layer supports 3 operations described in detail in Section 3. XMLP Operations are modelled as sequences of events crossing the layer boundary between XML Protocol Processors and XML Protocol Applications.

2. XML Protocol Abstract Model Overview.

Figure 2.1 below presents a simple case of the XML Protocol abstract model. Hosts I and V each contain XML protocol application components, which **use** the services of the XML protocol layer, and XML protocol layer components which **provide** the services of the XML protocol layer. The services of the XML protocol layer are abstracted at the upper layer boundary as a single operation, XMLP_UNITDATA which is described in detail in Section 3.



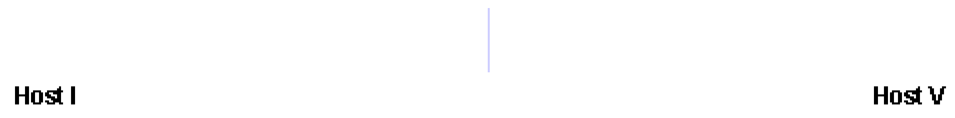


Figure 2.1 Model of Simple Case without Intermediaries.

Figure 2.2 below shows 5 hosts in a more complex case of the XML Protocol abstract model. Hosts I, III and V each contain XML protocol application components. Hosts II and IV are intermediaries that operate within underlying protocol layers such as HTTP proxies and SMTP message routers.

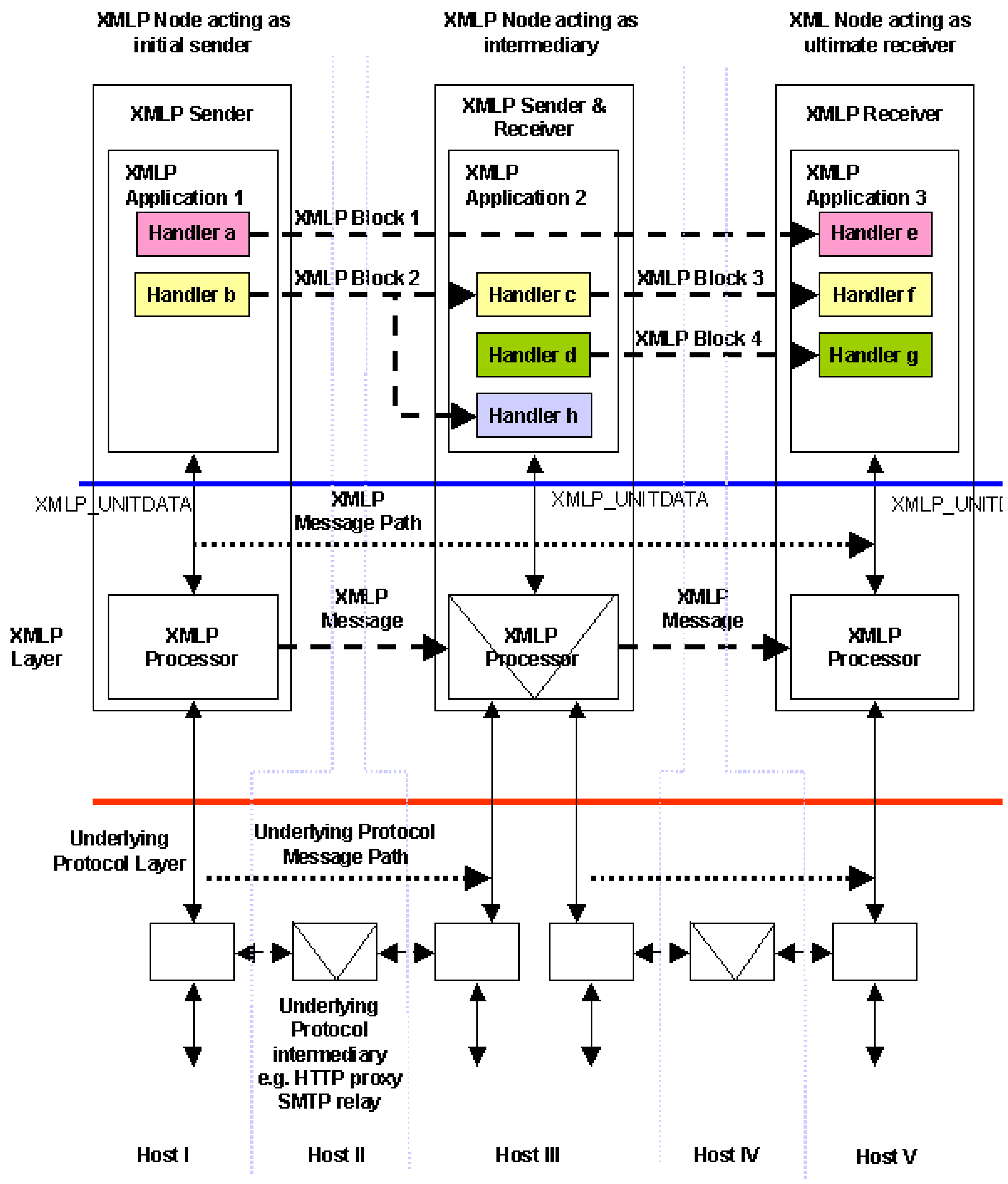


Figure 2.2 XML Protocol Model Overview

Figure 2.2 can be used to discuss a number of message exchange scenarios. For example, the XML protocol Processor at Host III is bound to two, possibly different, underlying protocols. It could serve merely as a 'helper' to transition an XML protocol message from one underlying protocol to another in circumstances where the initiating processor is bound to a different underlying protocol infrastructure than the receiving or responding node, say Host V in figure 2.2. A similar scenario arises if Host III is part of an XML Protocol Firewall that controls the ingress and egress of messages from a given organisation. In both these circumstances the XML Protocol Handler(s) within the XML protocol application at Host III need not be present.

If we turn our attention to the operation of the XML protocol applications above the XML protocol layer boundary, we have a scenario in which the application at Host I has some XML protocol blocks to deliver to Host V. In addition the application at Host I needs to trigger Intermediary functionality at Host III by the inclusion of several XML Protocol Blocks. "XML Protocol Block 1" is intended for "XML protocol handler (e) " within the application on Host V. Block 2 is intended for handler (h) and handler (c) which replaces Block 2 with Block 3. Also, the XML protocol application at Host III inserts Block 4 into the message forwarded from Host III to Host V. Blocks 3 and 4 are intended for handlers (f) and (g).

3 XML Protocol Layer Service Definition

This section focuses on the definition of an abstract interface between the XML protocol applications and the XML protocol layer. It needs to be remembered that the layer interface described in this section is abstract - its purpose is to enable description, not to constrain implementation.

The services provided by the XML protocol layer are modeled a single operation. XMLP_UNITDATA provides services to sending, intermediary and receiving XML protocol applications.

3.1. XMLP_UnitData Operation

XMLP_UnitData is a best effort one-way message transfer operation with message correlation. Multiple message transfer operations can be correlated in various ways to form message exchange patterns like request/response, and long-lived dialogs.

The XMLP_UnitData operation is modeled by four primitives (events). Each primitive models a transmission, reception or status event at interface between an XML protocol application and an XML protocol processor:

```
XMLP_UnitData.send( To, [ImmediateDestination], Message, [Correlation],
   [BindingContext]);

XMLP_UnitData.receive( [To], [From], Message, [Correlation],
   [BindingContext]]);

XMLP_UnitData.status( [From], Status, [BindingContext]);

XMLP_UnitData.forward( [ImmediateDestination], Message, [BindingContext]]);

All parameters are detailed in Section 3.2
```

Conceptually the XMLP_UnitData operation encapsulates the transmission of an XML protocol message from a sending XML protocol application to a receiving XML protocol application. The principal conceptual difference between sending and forwarding an XML protocol message is that, from a message correlation point of view, sending generates a new message whereas forwarding passes on an existing message. Conceptually the forwarded message is the same message as previously received although the action of intermediary processing may have changed the value of the message.

Figure 3.1 below illustrates the normal use of these primitive at the sending and receiving XML protocol applications.

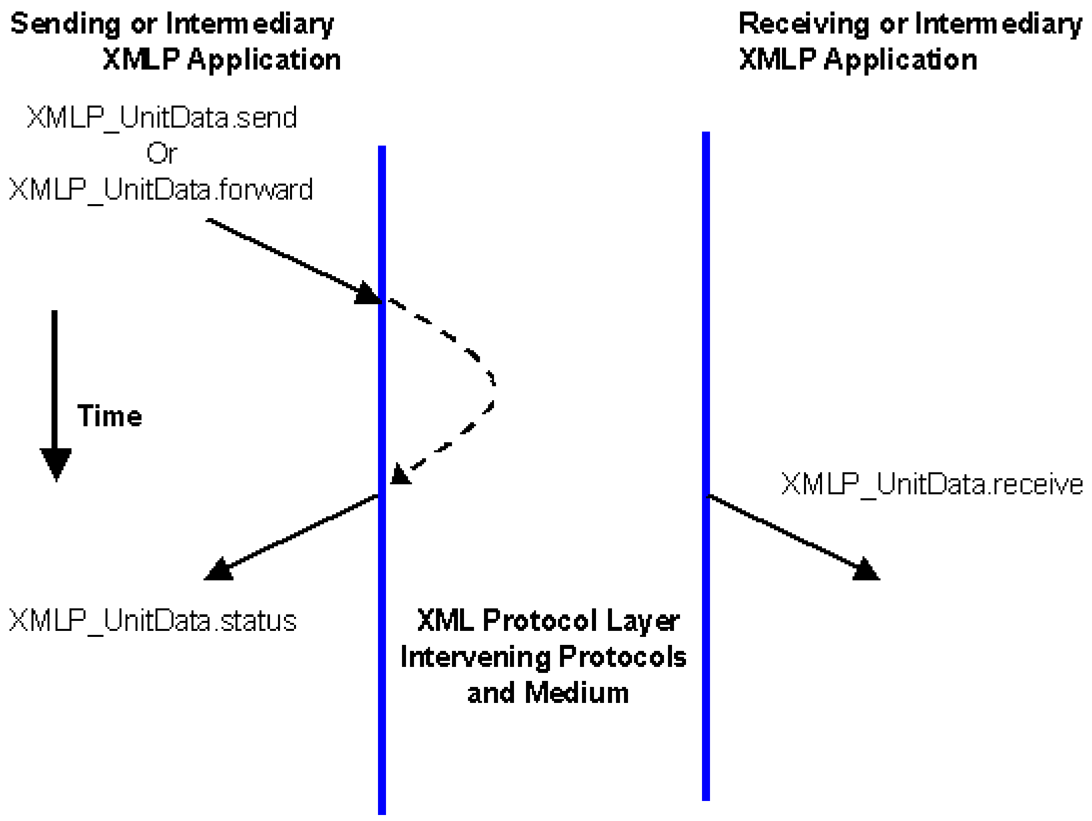

Sending or Intermediary XMLP Application
Receiving or Intermediary XMLP Application
XMLP_UnitData.send Or XMLP_UnitData.forward
Time
XMLP_UnitData.receive
XMLP_UnitData.status
XML Protocol Layer Intervening Protocols and Medium

Figure 3.1 XMLP_UNITDATA Operation

The operation is best effort which means that it can fail silently with the loss of the message in transit. A lost message may have been partially processed at an intermediary XML protocol application. The success or failure of the operation is reported via the XMLP_UnitData.status primitive. In some circumstances it may only be possible to report that a message has been sent. In other circumstances it may be possible to report that a message has or has not been delivered to its ultimate recipient.

XMLP_UnitData.send: Invoked by the sending XML protocol application and directed at the local sending XML protocol processor to start a one-way transfer operation.

Upon receipt of this primitive by the sending XML protocol processor an XML protocol message is transferred from the sending XML protocol processor toward the receiving XML protocol processor (possibly via intermediary XML protocol processors).

This primitive differs from the .forward primitive in that it is used by the initial sender of an XML protocol message to send a new message.

XMLP_UnitData.receive: Invoked by the receiving XML protocol processor and directed at a local receiving XML protocol application to deliver a received XML protocol message.

This primitive is invoked as a result of the arrival of an XML protocol message from the sending XML protocol processor (via the underlying protocol layers).

XMLP_UnitData.status: Used to report on the delivery status of the operation to the sending XML protocol application. This primitive may be used to report to the sending XML protocol application on the success or failure to send and deliver a message to the receiving XML protocol application. In general, it is not possible to assert that a message has been delivered to the receiving XML protocol application without engaging in further interactions. With care it is possible to assert definite failure to deliver provided that circumstances are such that there is no possibility of subsequent delivery. From the point-of-view of the initiating XML application the operation has completed once this primitive has been invoked.

XMLP_UnitData.forward: Invoked by an intermediary XML protocol application once it has completed intermediary processing of a message in transit and directed at the local intermediary XML protocol processor.

In the event of success the message is forwarded to its next destination, as designated by the ImmediateDestination parameter if given. Alternatively an implementation or configuration dependent method may be used to select the next recipient of the message along a path.

In the event of failure, the message in transit is discarded. A correlated fault message may be generated by the intermediary XML protocol application and sent toward the originator of the failed message.

This primitive differs from the .send primitive in that it is used by an intermediary XML protocol application to forward an existing XML protocol message received by the intermediary XML protocol application.

An XML protocol application may engage in multiple concurrent operations with the same or different intermediary and/or receiving XML protocol applications. These concurrent operations are independent and the order in which they are processed by the receiving and intermediary applications may be different from the order in which they are invoked or complete at the sending XML protocol application.

3.1.1 Correlation at Sending and Receiving XML Protocol Applications

The Correlation parameter provides a general mechanism by which richer message exchange patterns such as request-response and request/multi-response can be derived on top of the one-way message exchange pattern of the XMLP_UnitData operation. The mechanism by which correlation is determined is **not** specified in this abstract model.

Message correlation may be determined through:

- the exploitation of features in the underlying protocol eg. the request/response nature of HTTP;
- mechanism introduced either by the XMLP processor to operate across multiple possible underlying protocols.
- mechanism introduced by a binding to a particular underlying protocol within the domain of the underlying protocols own header extension mechanism.

When included in an XMLP_UnitData.send primitive Correlation.MessageRef indicates that the XML protocol message being sent is a direct consequence of the processing of an XML protocol message previously received by the sending XML protocol application and referenced locally by Correlation.MessageRef.

Likewise, when included in an XMLP_UnitData.receive primitive Correlation.MessageRef indicates that the message being received is a direct consequence of the processing of a XML protocol message previously sent by the receiving XML protocol application and referenced locally by

Correlation.MessageRef.

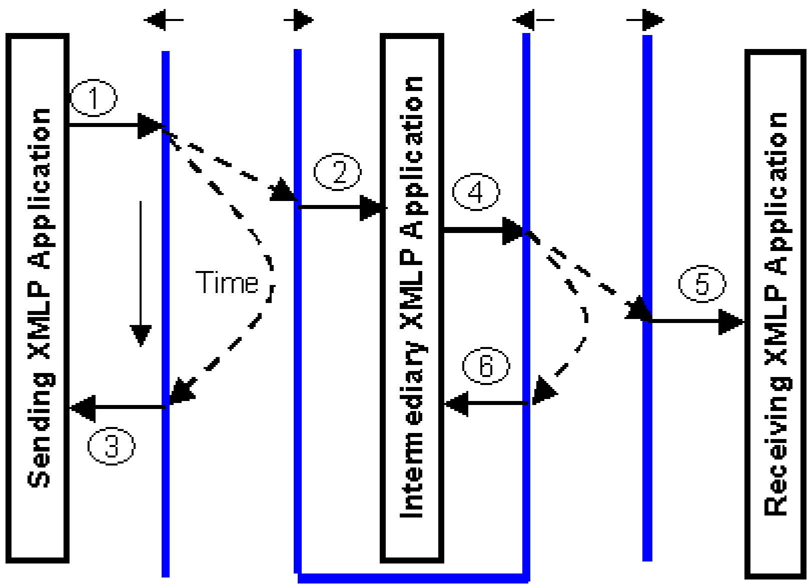
Failures that arise during message processing at the recipient or at intermediary XML protocol applications may result in the generation of fault messages directed toward the originator of the message whose processing gave rise to the fault. Such fault messages are a direct consequence of the faulted message and this should be indicated through the use of the Correlation parameter.

3.1.2 XMLP_UnitData Operation through Intermediaries

Conceptually an XML protocol intermediary does not generate a new XML protocol message, it operates on an XML protocol message in transit. Thus the received message and the forwarded message are regarded as the same message although the intermediary may change the value of the message.

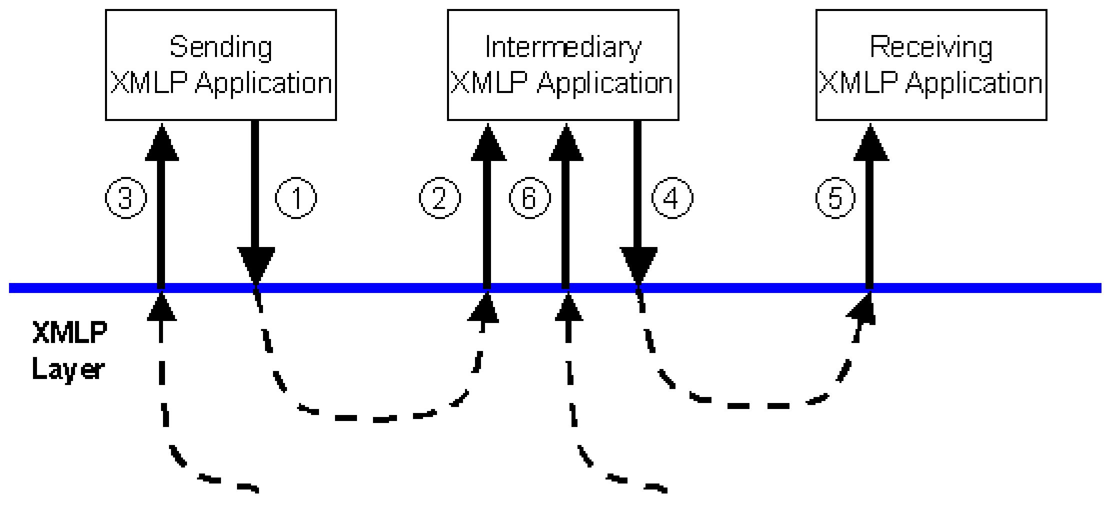
Figure 3.2 shows the normal behaviour of an XML_UnitData operation through an intermediary in the absence of fatal failures. The three vertical lines represent the local XML protocol layer boundaries and the small arrows above denote the up/down orientation of the boundary. Figure 3.3 below shows an alternate representation of the same scenario.

The scenario depicted in figures 3.2and 3.3. show just a single intermediary interposed in the operation however the principle extends to an arbitrary number of intermediaries.



Layer Primitives Key
1. XMLP_UnitData.send
2. XMLP_UnitData.receive
3. XMLP_UnitData.status (any time after 1.)
4. XMLP_UnitData.forward
5. XMLP_UnitData.receive
6. XMLP_UnitData.status (anytime after 4)

Figure 3.2 Normal XMLP_UnitData operation through an Intermediary



Layer Primitives Key

1. XMLP_UnitData.send
2. XMLP_UnitData.receive
3. XMLP_UnitData.status (any time after 1)
4. XMLP_UnitData.forward
5. XMLP_UnitData.receive
6. XMLP_UnitData.status (any time after 4)

Numerical ordering indicates time sequence

Figure 3.3 Normal XMLP_UnitData operation through and Intermediary (alternate treatment)

It is worth noting that the XMLP_UnitData.status is generated from within the XML protocol layer. It may indicate anything from the mere fact that the message has been sent or forwarded by the sending node; that its has been received and/or sent from the intermediary node; or that it has indeed been delivered to the ultimate recipient node. What it means in a given circumstance will depend upon the capabilities of the underlying communications protocols used to construct the message path. The strongest thing that it can indicate is the failure to deliver an XML protocol message to its ultimate recipient.

3.1.3 Message Correlation at Intermediary XML Protocol Applications

The Correlation.MessageRef sub-field of the optional Correlation parameter on a XMLP_UnitData.receive primitive carries a local abstract reference to an XML protocol message that was previously forwarded by this intermediary XML protocol application. The current message is a direct consequence of the processing of that earlier forwarded message.

Typically this will arise when an application level response travels along a path that passes through one or more of the same intermediary XML protocol applications that the corresponding request passed through earlier.

3.2 Operation Parameters

This section describes the operation parameters used in the operation primitives described above.

To	An abstract reference that denotes the XML protocol application that a

	message was originally sent to by the initiating or sending XML protocol application.
From	An abstract reference denotes the sending XML protocol application in *.receive* primitives. In *.receive* primitives this parameter makes the identity of the sending/initiating XML protocol application available to the receiving/responding XML protocol application. *[Intermediaries may obscure this or we may require that they don't... discuss!]* In the *XMLP_UnitData.status* primitive, this parameter conveys the identity of the XML protocol application to which an XML protocol message was sent after any redirections imposed by underlying protocols. NB. Further redirections may occur that cannot be reported. *I[Again possibly obscured by intermediaries...]*
ImmediateDestination	An identifier that denotes the immediate destination of an XML protocol message. If this parameter is unspecified, the default value is implementation and configuration dependent. This parameter enables sending and intermediary XML protocol applications to address the message to the next intermediary on route.
Message	An abstraction of an XML protocol message exchanged between sending and receiving XML protocol applications. An XML protocol message has the following sub-fields: Message.Faults; Message.Blocks; and Message.Attachments.
Message.Faults	An abstraction of a collection of XML protocol faults carried in an XML protocol message that is correlated with the XML protocol message whose processing gave rise to one or more faults. Such a message may arise at an intermediary or at the ultimate recipient.
Message.Blocks	An abstraction of the XML protocol blocks within an XML protocol message which are intended to be processed by intermediaries or the ultimate recipient.
Message.Attachments	An abstraction of the a collection of arbitrary attachments being transferred as part of an XML protocol message. These attachments are opaque bytes as far as XML protocol processing elements are

	concerned From the point-of-view of abstract service definition the actual mechanism used to transfer attachments is immaterial, however particular bindings may employ more efficient mechanisms than others. *[NB. This places an obligation on XML protocol binding specifications to specify how attachments are to be carried.]*
Correlation	An optional parameter used to express local relationships between XML protocol messages. At present only a single subfield, Correlation.MessageRef is defined, however it is conceivable that other subfields may be defined in future, eg. Correlation.MsgSequence to distinguish between and potentially order n multiple messages that arise from the same source as a direct consequence of the current message.
Correlation.MessageRef	An abstraction of a local reference to the local abstraction of an XML protocol message the processing of which the current XML protocol message is a direct consequence. In XMLP_UnitData.send primitives, the value of this parameter references an XML protocol message previously received by the sending XML protocol application. In XMLP_UnitData.receive primitives, the value of this parameter references an XML protocol message previously sent or forwarded by the receiving application.
BindingContext	This parameter references an abstract structure that carries information pertinent to the underlying protocol binding(s). For example it may carry certificates, ids and passwords to be used by the sending/initiating XML protocol application to authenticate itself and/or to establish a secure channel. At the responding XML protocol application it may carry the authenticated id of the principal on whose behalf the operation is being conducted. If the information present in the BindingContext is inadequate for the execution of a given service primitive the invocation of that primitive will fail with a result that indicates why progress was not possible. BindingContext is optional and if not supplied the local default binding will be used. In the case of multiple bindings being available, inbound BindingContext indicates how an inbound message was received and outbound BindingContext constrains the choice of binding used for a given operation

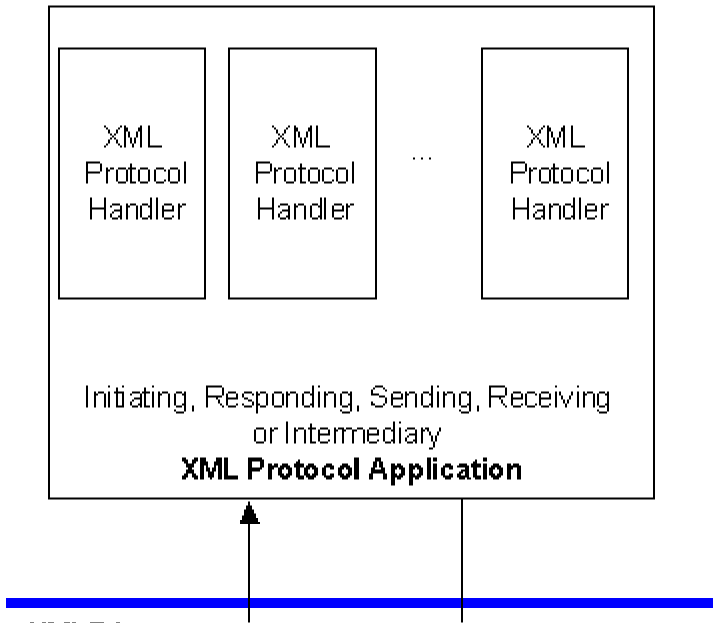
	given operation. BindingContext is discussed further in Section 5.2. *[NB This concept places another obligation on XML protocol binding specifications in that they must enumerate what binding specific information they require in an outbound BindingContext and what binding specific information they provide in inbound BindingContexts.]*
Status	In .status primitives this parameter indicates the disposition of the request operation which may be: *MessageSent, MessageDelivered, Unknown* and *FailedAtIntermediary*. The interpretation of a status value may be augmented by information carried in the *BindingContext*.

4. XML Protocol Applications and Modules

There is a significant debate over terms and concepts around Modules, Handlers, Targeting and Message routing. At the time of this draft the discussion is still open and this section will be updated to reflect any consensus arising from that discussion.

An XML protocol application is the logic at an XML processor that makes use of the core messaging services of the XML protocol. XML protocol applications may initiate, respond or act as intermediaries in XML protocol operations. Logically, an XML protocol application contains a number of XML protocol handlers that are responsible for applying the processing rules associated with XML protocol modules. The unit of exchange between XML protocol handlers are XML protocol blocks.

XML protocol blocks are aggregated into XML protocol messages and may be targeted at particular XML processors (see Section 4.2). XML protocol blocks are delivered together with the rest of the XML protocol message which encapsulates them (and its attachments if any) to the targeted XML processor. The XML protocol application is then responsible for identifying and dispatching the appropriate XML protocol handlers. Generally, the dispatch to a handler will be determined by the presence of an associated block or blocks, but not necessarily. Handler d in Figure 2.2 illustrates such a case.



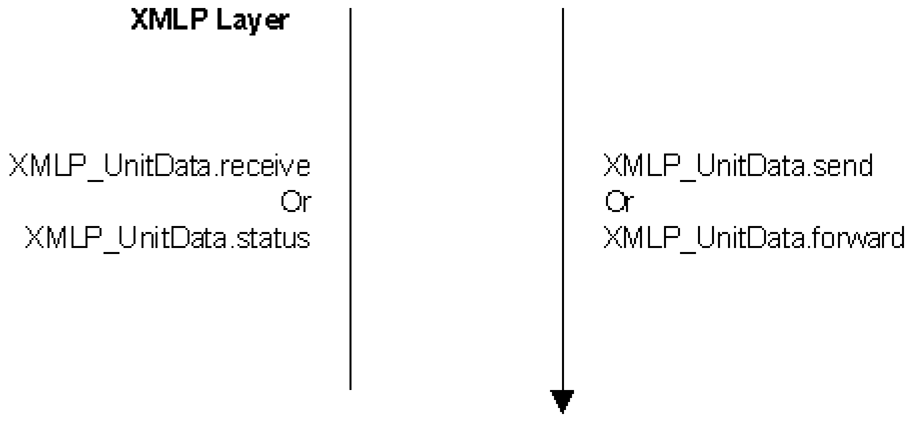


Figure 4.1 XML Protocol Application

Each handler may succeed or fail fatally. It is the responsibility of the XML protocol application to determine the overall result of the actions of any XML protocol handlers it invokes and to augment any Faults structure carried in the ongoing message. In cases where there are multiple influences on the ImmediateDestination, it is also the responsibility of the XML protocol application to resolve any conflicts.

4.1 XML Protocol Message Routing and Targeting (aka Naming and Addressing :-))

Needs to use some terms here that arise from the intermediaries thread Martin started

An XML protocol message path can be viewed as the sequence of handlers that an XML protocol message passes through between initiating/sending XML protocol application and receiving responding XML protocol application. With reference to figure 2.2, the diagram in figure 4.2 depicts the message path of the corresponding XML protocol message under the XML_UnitData operation.

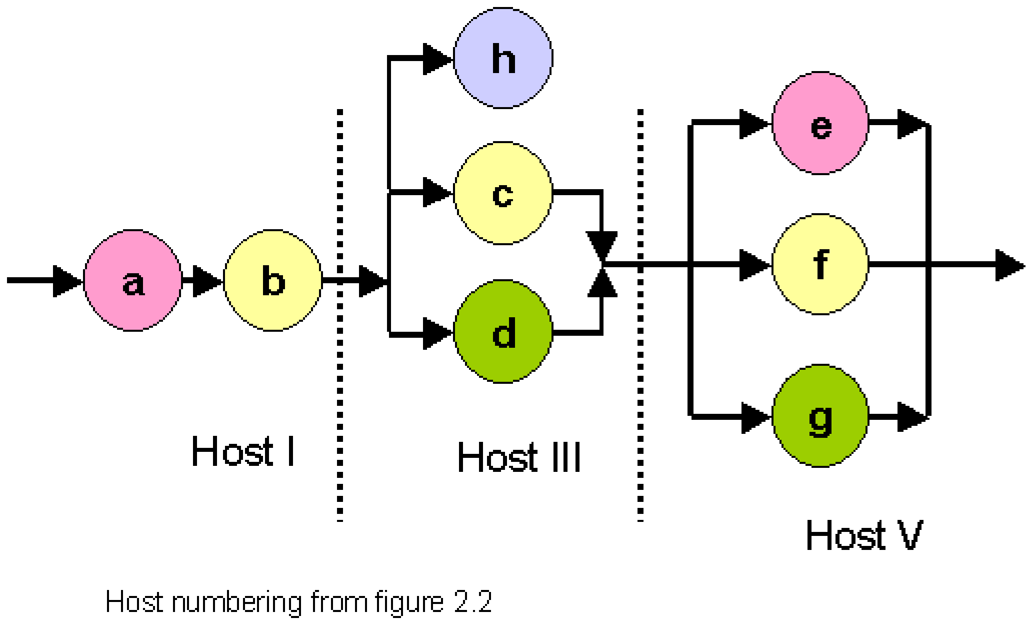


Figure 4.2 XML Protocol Message Path

The path in figure 4.2 shows sequential handler processing at the sending node, Node I, while the handler processing at Nodes III and V is concurrent (at least logically). Combinations of handlers that can be invoked concurrently from within an XML protocol application are said to be mutually orthogonal.

4.2 XML Protocol Modules and Message Processing

XML protocol modules are the unit of extension within the XML protocol. An XML protocol module encapsulates the syntactic constructs of an extension, known as XML protocol blocks, and the behavioural rules associated with the generation and processing of an XML protocol block. The abstraction for the processing and/or logic defined by an XML protocol module is called an XML protocol handler.

The SOAP 1.1 specification (section 2) states: "Processing a message or a part of a message requires that the SOAP processor understands, among other things, the exchange pattern being used (one way, request-response, multicast, etc.), the role of the recipient in that pattern, the employment (if any) of RPC mechanisms such as the one documented in section 7, the representation or encoding of data, as well as other semantics necessary for correct processing." An XML protocol module is the locus for understanding blocks associated with that module. A given message may employ the services of many modules, both generic (e.g., security, caching, compression, transactions, etc.) and application-specific.

The following list provides an initial set of concepts which capture and slightly refine the SOAP message processing model. A comparison of each concept with SOAP is also provided for reference.

1. An XML Protocol message consists of a set of zero or more blocks.

   SOAP: Similar. Blocks correspond to header or body entries. SOAP groups header entries into an optional Header element and body entries into an obligatory Body element.

2. Each block has the following sub-fields: **Block.Id**, **Block.Actor**, and **Block.MustUnderstand**. Block.Id is an optional identifier that identifies the block for the purposes of reference by other blocks. Block.Actor identifies the XMLP processor that is intended to process the block. Block.MustUnderstand specifies whether the intended semantics of the block must be carried out.

   SOAP: SOAP does not specify whether an actor URI is to be interpreted extensionally (naming a particular node) or intensionally (describing a node or group of nodes that satisfy some property). Special reserved URI's describe nodes which are encountered next or last. Beyond the reserved URI's, there is no particular semantics associated with an actor URI. Semantically, the URI's can signify a processor that supports a given application, module or capability, or it can describe a destination, node or location. This flexibility is preserved in XMLP.

3. The fully qualified name of the top element of a block identifies the block.

   SOAP: SOAP identifies blocks by the fully qualified element name. The block can (but need not) be mapped to some appropriate handler. Other schemes have also been suggested. For example, an attribute could name a module which would take responsibility for selecting the handler to invoke.

4. The following values for Block.Actor have special significance: **Next**, **Final**, and **None**. Next matches the next processor. Final matches the final processor. None is for untargeted blocks which may be referenced by other blocks.

   An empty actor defaults to Final. An untargeted block marked with `None` is useful for declarative information that is referenced by another block or blocks. It is guaranteed not to be removed and can even be referenced by blocks which are targeted at different processors.

SOAP: An empty SOAP actor in a header "indicates that the recipient is the ultimate destination of the SOAP message," and a "body entry is semantically equivalent to a header entry intended for the default actor." This is what Final designates. The intended (final) processor must recognize itself as such. Next has the same interpretation as the SOAP URI, `http://schemas.xmlsoap.org/soap/actor/next`. SOAP forces the actor for body entries to be the final processor. SOAP permits the inclusion of blocks for which there do not turn out to be any actors that match along the message path; and even if an actor URI matches a given processor, the processor may determine that no behaviour is associated with the block. The value None, on the other hand, is a stronger statement on the part of the sender that signifies that no processor will qualify as a matching actor.

5. When a block is selected for processing at an intermediary, the block is removed from the envelope. A handler may add zero or more blocks. Blocks which are merely referenced are not removed.

   SOAP: SOAP doesn't allow body entries to be processed at intermediaries and hence they are never removed at an intermediary.

6. The XML Protocol blocks are ordered within the envelope. This order is followed by each processor as it selects and processes blocks, yielding a limited facility for specifying sequential constraints. Two alternatives are available for more complex orderings and constraints. Hierarchical constraints can be achieved by syntactically scoping blocks inside one another. Finally, blocks can be incorporated by reference using the "`id`" and "`href`" attribute mechanism. Using these techniques, more elaborate "manifest" blocks which direct the processing of other blocks can be designed. From the processor's point of view, only the outermost element of the block is seen.

   SOAP: Header entries can be referenced via links from other headers. If they have no actor (targeted at the final destination), they will not be removed by any intermediaries. Using that mechanism, headers can be effectively shared among modules, even at different nodes. The actor-less headers are interpreted as relevant to the final processor, even though they may not be. The body can only be targeted at the final procesor.

7. The processing of a block by a handler may result in a fault or a successful evaluation. A fault terminates processing of the block and message and causes a return message containing the fault to be generated if a return path is available. Rather than fatally faulting, it is also possible for a handler to insert a block targeted to another destination e.g., the final destination). This block can contain status information, non-fatal errors, or other results that can be further processed, incorporated into a return value, etc.

   SOAP: Similar.

5. Underlying Protocol Bindings

It is the intent that the XML protocol be capable of being bound to a variety of underlying communication protocols. The XML protocol working group will define a binding to HTTP. It is anticipated that others will create bindings to SMTP, TCP, SSL, BEEP and others.

5.1 Binding Service Model

This entire subsection is new and has not be subject of significant debate. It should be regarded as a work in progress.

5.1.1. Introduction

This section presents an abstract service model that XML protocol bindings will supply to the upper XML protocol layer. The intent is to describe the interactions between the XML protocol processor and underlying protocol bindings and to demonstrate how these interactions are choreographed to enable multiple message exchange patterns. This model is intended to provide a framework in which the development of concrete binding specifications can be discussed. This is not intended as an API specification.

The diagram below shows a logical layered view of the binding model with the XML protocol processor being bound to four underlying transports.

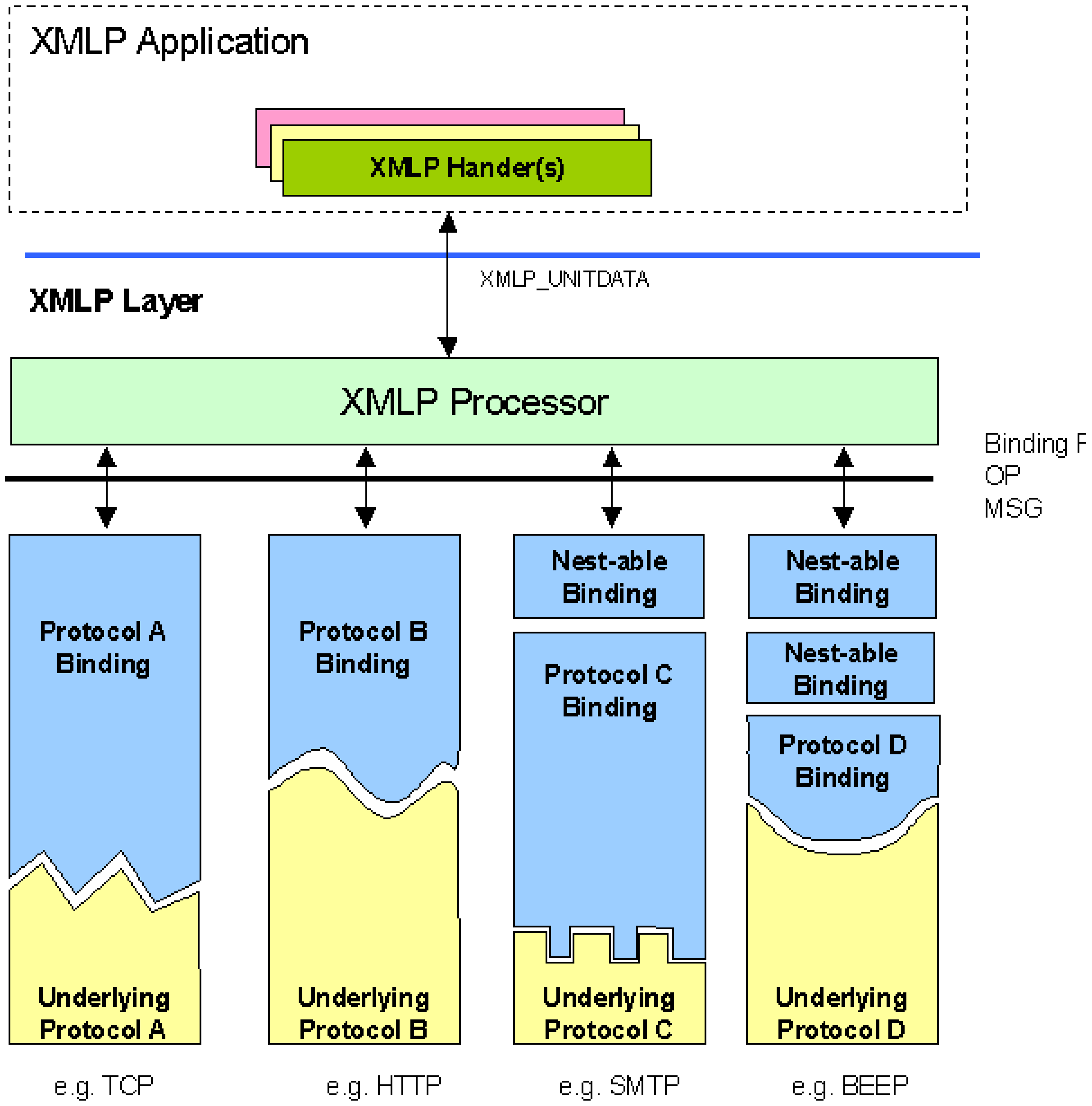


Figure 5.1 Binding Model

This document concerns itself with the interactions at the solid black line between the XML protocol processor and a given binding.

Note that, as shown, some bindings may be nested. e.g. a MIME binding might be nested within a HTTP binding to allow additional binary data to be sent along with (but outside) the XMLP envelope.

5.1.2. Service Primitives

5.1.2.1 Message Exchange

There are two primitives associated with message exchange: MSG.req and MSG.ind. A MSG.req primitive is sent from the XML protocol processor to the binding in order to cause the binding to send a message. A MSG.ind primitive is sent from the binding to the XML protocol processor to indicate arrival of a message.

5.1.2.2 Message Correlation

In order to support message exchange patterns that are more complex than the simplest one-way exchange, some form of message correlation is required. For example, in a request-response message exchange there must be some means of correlating the request with the response. In this document a single instance of a message exchange pattern is referred to as an XML protocol processor operation or just operation for short.

There are four pairs of primitives associated with operation delineation and hence message correlation:

1. OP.start-req and OP.start-conf

   A OP.start-req primitive is sent from the XML protocol processor to the binding to request initialisation of a new correlated message exchange. The binding responds with a OP.start-conf primitive.

2. OP.start-ind and OP.start-resp

   A OP.start-ind primitive is sent from the binding to the XMPL layer to indicate that a new correlated message exchange is being requested. The XML protocol processor responds with a OP.start-resp primitive.

3. OP.end-req and OP.end-conf

   An OP.end-req primitive is sent from the XML protocol processor to the binding to terminate a correlated message exchange. The binding responds with an OP.end-confprimitive. Whilst the OP.end-conf is outstanding, the XML protocol processor must be prepared to continue to receive MSG.inds

4. OP.end-ind and OP.end-resp

   An OP.end-ind primitive is sent from the binding to the XML protocol processor to indicate that a correlated message exchange is to be terminated. No further MSG.ind will be delivered as part of the corresponding operation, however the XML protocol processor receiving the OP.end-ind primitive may continue to issue MSG.req primitives to complete operation in progress. Once all MSG.req primitives associate with the operation have been issued the XML protocol processor concludes the operation by with the invocation of an OP.end-resp primitive."

The actual correlation mechanism is underlying protocol and implementation specific. e.g. A OP.start-conf may carry some unique identifier that must be provided with any subsequent MSG.req (s) and is included with any subsequent MSG.ind(s).

There is no retained state within the binding between operations although there may be during operations.

5.1.2.3 F

5.1.2.3 Errors

The final primitive ERR.ind is sent from the binding to the XML protocol processor when an error occurs, e.g. if a MSG.req cannot be honoured then an ERR.ind is generated. Errors are correlated to a particular message exchange using the mechanism described above.

5.1.3. Message Exchange Patterns

The following sections illustrate the choreography of XML protocol binding primitives for a number of different message exchange patterns. These are intended to be illustrative rather than proscriptive. In particular, in many cases either sender or receiver might initiate the OP.end-req.

5.1.3.1 One Way Message

Sender

OP.start-req, OP.start-conf, MSG.req, OP.end-req, OP.end-conf.

Receiver

OP.start-ind, OP.start-resp, MSG.ind, OP.end-ind, OP.end-resp.

Comments

Note that depending on the underlying protocol the primitives at sender and receiver may not operate in lock-step. In particular, the OP.start-ind may not be delivered to the receiving XML protocol processor until the sending XML protocol processor has issued the MSG.req or even the OP.end-req. An alternative way of saying this is that a binding may choose to delay making an underlying protocol connection until a message needs to be sent.

5.1.3.2 Request Response

Sender

OP.start-req, OP.start-conf, MSG.req, MSG.ind, OP.end-req, OP.end-conf.

Receiver

OP.start-ind, OP.start-resp, MSG.ind, MSG.req, OP.end-ind, OP.end-resp.

5.1.3.3 Request and n Responses

Sender

OP.start-req, OP.start-conf, MSG.req, MSG.ind, MSG.ind, ..., OP.end-ind, OP.end-resp.

Receiver

OP.start-ind, OP.start-resp, MSG.ind, MSG.req, MSG.req, ..., OP.end-req, OP.end-conf.

5.1.4. Sample Mappings

5.1.4.1 HTTP

The following tables show how the binding primitives might map onto the HTTP protocol actions on

The following tables show how the binding primitives might map onto the HTTP protocol actions on the initiator and responder for a request-response message exchange, time increases moving down the tables.

Initiator

Binding Primitive	Binding Action
OP.start-req	
OP.start-conf	
MSG.req	
	Send POST request
	Receive POST results
MSG.ind	
OP.end-req	
OP.end-conf	

Responder

Binding Primitive	Binding Action
	Receive POST request
OP.start-ind	
OP.start-resp	
MSG.ind	
MSG.req	
	Send POST results
OP.end-ind	
OP.end-resp	

The above assumes use of HTTP persistent connections.

5.1.4.2 SMTP

The following tables show how the binding primitives might map onto the SMTP protocol actions on the initiator and receiver for a simple one-way message exchange, time increases moving down the tables.

Initiator

Binding Primitive	Binding Action
OP.start-req	
	Open SMTP session
OP.start-conf	
MSG.req	
	Send mail message
OP.end-req	
	Close SMTP session
OP.end-conf	

Responder

Binding Primitive	Binding Action
	Begin SMTP transaction

	Begin SMTP transaction
	Receive mail message
	End SMTP transaction
OP.start-ind	
OP.start-resp	
MSG.ind	
OP.end-ind	
OP.end-resp	

5.1.5. Binding Considerations

Underlying protocols may provide various levels of functionality to the binding. It is the responsibility of the binding to implement a mapping between XML protocol service primitives and underlying protocol primitives. The mapping should make the best use of the facilities of the underlying protocol and maximise efficiency where possible, e.g. connection setup is generally an expensive operation - bindings for connection oriented protocols should attempt to minimise the number of connections made for a given message exchange pattern. In particular, when defining a mapping the following need to be specified:

Protocol
The binding should identify the exact protocol to which XML protocol is being bound including a version. Examples might be HTTP/1.1 or SMTP[RFC821].

Addressing
The binding needs to show how to specify an XML protocol processor's address with an URL.

Message Passing

The binding needs to specify unambiguously how to use the underlying protocol to pass a whole XML Protocol message to a node specified by a given address. Depending on the underlying protocol capabilities, the specification may need to detail the following:

1. Use of underlying protocol primitives for sending and receiving messages.
2. Use of underlying protocol headings.
3. Underlying protocol connection management including roles of initiator and responder, how to handle abnormal terminations, can responder terminate connection, etc.

Message Exchange Pattern(s)
The binding needs to specify how underlying protocol sessions are used in common message exchange patterns including one-way and request-response.

[Question: what other message exchange patterns should we specify here ?]

Here, protocol session means a unit of communication in the underlying protocol, in HTTP this maps to a single request/response, in SMTP a session only covers a single act of sending a message or a single act of receiving a message. In BEEP the session would possibly map to a channel that would be capable of many different message exchange patterns.

Message Ordering Characteristics
The binding needs to specify what message ordering characteristics the underlying protocol supports. e.g. If two messages are sent in the same direction in the same session is their order of arrival guaranteed to be the same as the order in which they were sent.

Error Handling
The binding needs to specify how errors in the underlying protocol will be handled. A non-exhaustive list of things to consider here is: connection errors, addressing errors, message transmission errors, abnormal termination.

[Question: what other types of error do we need to consider ?]

5.2 BindingContext

Each of these underlying protocols supports different features and capabilities and it is not plausible or desirable to provide a detailed abstraction that captures the full range of diversity. The core of XML protocol in respect of the exchange of XML protocol messages takes a lowest common denominator approach by regarding the underlying channel as potential lossy and capable of mis-ordering and duplication. Underlying protocols may offer better assurances of delivery probability, delivery ordering and at-most once delivery behaviour.

In the service abstraction provided above, an abstract parameter known as BindingContext is introduced. The primary purpose of BindingContext is to act as a collecting 'bucket' for parameters that control the functionality of the particular set of underlying protocols available at any given node.

It is expected that the authors of XML Protocol binding specifications will add structure beneath BindingContext to cover the features and capabilities of the underlying protocol being bound. This may also include a descriptor of the ordering, loss and duplication properties of the underlying protocol, although this should be treated with caution in multi-hop scenarios.

Some BindingContext extensions may be of more general applicability than just a single binding. For example, the references to user ids, private keys and public certificates necessary for SSL and HTTPS could be shared between both bindings (were they to exist).

One would therefore expect the structure under BindingContext to grow along the lines of:

```
BindingContext.Shared    //A substructure for information shared by
several bindings
BindingContext.HTTP      //A substructure for information related to
HTTP
BindingContext.SMTP      //....

....and so on.
```

The manipulation of fields within the BindingContext may be driven from within, for example, an intermediary XML protocol application on the basis of constructs carried as XML protocol message blocks within the message being carried.

Hopefully this captures the general idea behind BindingContext... the details will evolve over time... indeed they will evolve as bindings get described.

5.3 Attachment of Arbitrary Content

*This topic is subject to active discussion and the view presented here is *very* preliminary. There is likely be considerable diversity of viewpoints that are not captured let alone resolved here.*

Another role of an XML protocol binding is to invoke the services of underlying protocols and to introduce any mechanism required to map between the semantics of the underlying protocol and those of the XML protocol core message delivery operations XMLP_UnitData. The attachment of arbitrary content to an XML protocol message is one facet of this mapping.

The core XML protocol messaging services intrinsically handle arbitrary attachments through the use of the Attachments parameter. The expectation is that the design of XML protocol WILL specify a means for encoding arbitrary content and carrying it within an XML protocol envelope. This mechanism will leverage any pre-existing work within XML Schema, and will also provide mechanisms for embedding complete, arbitrary, XML documents within the outer XML protocol message envelope (itself an XML construct).

Some underlying protocols will support more efficient ways of carrying arbitrary content and or multiple XML documents. The normative bindings to an underlying protocol MUST define the mechanism used by that binding to carry attachments containing arbitrary content. In the absence of any statement to the contrary in the definition of a particular protocol binding, the default XML based encoding for arbitrary content attachments will be taken as having been specified. Any other scheme specified for a particular binding must have functional capabilities at least as capable as the default XML based encoding scheme, in particular it must be possible to reference the individual attachments from within the XML protocol message envelope.

6. References

[XMLPReqs] "XML Protocol (XMLP) Requirements" http://www.w3.org/TR/2001/WD-xmlp-reqs-20010319/#N2082

[SOAP 1.1] "Simple Object Access Protocol (SOAP) 1.1" http://www.w3.org/TR/SOAP/

[Issues] "XML Protocol WG Issues List" http://www.w3.org/2000/xp/Group/xmlp-issues.html

7. Acknowledgements

This document is the work of the W3C XML Protocol Working Group.

Members of the Working Group are (at the time of writing, and by alphabetical order): Yasser al Safadi (Philips Research), Vidur Apparao (Netscape), Don Box (DevelopMentor), David Burdett (Commerce One), Charles Campbell (Informix Software), Alex Ceponkus (Bowstreet), Michael Champion (Software AG), David Clay (Oracle), Ugo Corda (Xerox), Paul Cotton (Microsoft Corporation), Ron Daniel (Interwoven), Glen Daniels (Allaire), Doug Davis (IBM), Ray Denenberg (Library of Congress), Paul Denning (MITRE Corporation), Frank DeRose (TIBCO Software, Inc.), Brian Eisenberg (Data Channel), David Ezell (Hewlett-Packard), James Falek (TIBCO Software, Inc.), David Fallside (IBM), Chris Ferris (Sun Microsystems), Daniela Florescu (Propel), Dan Frantz (BEA Systems), Dietmar Gaertner (Software AG), Scott Golubock (Epicentric), Rich Greenfield (Library of Congress), Martin Gudgin (Develop Mentor), Hugo Haas (W3C), Marc Hadley (Sun Microsystems), Mark Hale (Interwoven), Randy Hall (Intel), Gerd Hoelzing (SAP AG), Oisin Hurley (IONA Technologies), Yin-Leng Husband (Compaq), John Ibbotson (IBM), Ryuji Inoue (Matsushita Electric Industrial Co., Ltd.), Scott Isaacson (Novell, Inc.), Kazunori Iwasa (Fujitsu Software Corporation), Murali Janakiraman (Rogue Wave), Mario Jeckle (Daimler-Chrysler Research and Technology), Eric Jenkins (Engenia Software), Mark Jones (AT&T), Jay Kasi (Commerce One), Jeffrey Kay (Engenia Software), Richard Koo (Vitria Technology Inc.), Jacek Kopecky (IDOOX s.r.o.), Alan Kropp (Epicentric), Yves Lafon (W3C), Tony Lee (Vitria Technology Inc.), Michah Lerner (AT&T), Richard Martin (Active Data Exchange), Noah Mendelsohn (Lotus Development), Nilo Mitra (Ericsson Research Canada), Jean-Jacques Moreau (Canon), Masahiko Narita (Fujitsu Software Corporation), Mark Needleman (Data Research Associates), Eric Newcomer (IONA Technologies), Henrik Frystyk Nielsen (Microsoft Corporation), Mark Nottingham (Akamai Technologies), David Orchard (JamCracker), Kevin Perkins (Compaq), Jags Ramnaryan (BEA Systems), Andreas Riegg (Daimler-Chrysler Research and Technology), Hervé Ruellan (Canon), Marwan Sabbouh (MITRE Corporation), Shane Sesta (Active Data Exchange), Miroslav Simek (IDOOX s.r.o.), Simeon Simeonov (Allaire), Nick Smilonich (Unisys), Soumitro Tagore (Informix Software), James Tauber (Bowstreet), Lynne Thompson (Unisys), Patrick Thompson (Rogue Wave), Randy Waldrop (WebMethods), Ray Whitmer (Netscape), Volker Wiechers (SAP AG), Stuart Williams (Hewlett-Packard), Amr Yassin (Philips Research) and Dick Brooks (Group 8760). *Previous members were*: Eric Fedok (Active Data Exchange) Susan Yee (Active Data Exchange) Alex Milowski (Lexica), Bill Anderson (Xerox), Ed Mooney (Sun Microsystems), Mary Holstege (Calico Commerce), Rekha Nagarajan (Calico Commerce), John Evdemon (XML Solutions), Kevin Mitchell (XML Solutions), Yan Xu (DataChannel) Mike Dierken (DataChannel) Julian Kumar (Epicentric) Miles Chaston (Epicentric) Bjoern Heckel (Epicentric) Dean Moses (Epicentric) Michael Freeman (Engenia Software) Jim Hughes (Fujitsu Software Corporation) Francisco Cubera (IBM), Murray Maloney (Commerce One), Krishna Sankar (Cisco), Steve Hole (MessagingDirect Ltd.) John-Paul Sicotte (MessagingDirect Ltd.) Vilhelm Rosenqvist (NCR) Lew Shannon (NCR) Henry Lowe (OMG) Jim Trezzo (Oracle) Peter Lecuyer (Progress Software) Andrew Eisenberg (Progress Software) David Cleary (Progress Software) George Scott (Tradia Inc.) Erin Hoffman (Tradia Inc.)

(Progress Software) David Cleary (Progress Software) George Scott (Tradia Inc.) Erin Hoffman (Tradia Inc.) Conleth O'Connell (Vignette) Waqar Sadiq (Vitria Technology Inc.) Tom Breuel (Xerox) David Webber (XMLGlobal Technologies) Matthew MacKenzie (XMLGlobal Technologies) and Mark Baker (Sun Microsystems).

We also wish to thank all the people who have contributed to discussions on xml-dist-app@w3.org.

8. Change Log

- Changes from Draft of 16th February 2001
    1. Added Mark Jones to list of contributors
    2. Removed Section 7 (Security)
    3. Renumbered Sections 5 and 6 as 4 and 5 respectively
    4. Section 1.1: Removed definitions covered by requirements document glossary and added reference to same.
    5. Updated Fig 2.1 (later renumbered Fig 2.2) with derivative of diagram in [XMLPReqs] (added service primitive annotations back in).
    6. Editorial on text section 2 to bring in line with revised Fig 2.1 (later renumbered Fig 2.2)
    7. Updated Figure 3.1 (was Figure 3.2) in response to Jacek's comment at F2F on how to indicate indeterminate ordering.
    8. Updated Figure 4.2 (was 5.2) to reflect changes in Figure 2.1 (later renumbered Figure 2.2; addition of handler h).
    9. Added Marc Hadley Binding Model as section 5.1 (and subsections) minor edits ("this document"->"this section")
    10. Added Mark Jones Module Processing Model as section 4.1.1.
    11. Forced case consistency on "XML protocol" application, processor and layer throughout (ish).
- Changes from Draft of 21st March 2001
    1. Updated section 4.1 from Mark Jones (promoted section 4.1.1 into 4.1)
    2. Corrected Numbering in References section 5-> section 6 (thanks Jacek!)
    3. Updated Figure 3.3 (was Figure 3.7) in response to another F2F comment from Jacek
- Changes from Draft of 26th March 2001
    1. Replaced section 3 with alternate version posted in http://lists.w3.org/Archives/Public/xml-dist-app/2001Mar/0229.html
    2. Replaced section 5.1 with new text from Marc Hadley addressing Hugo's comments in: http://lists.w3.org/Archives/Public/xml-dist-app/2001Mar/0221.html
    3. Pruned ToDo's list above.
    4. Pruned Issue's list.
- Changes from Draft of 27th March 2001
    1. Restructured issues list and removed resolved issues.
    2. Incorporated Henrik's feedback [HFN1]-[HFN29]
    3. Added Figure 2.2 (later renumbered Figure 2.1) in response to Issue 5 on previous draft
    4. Replaced section 4.1 (now section 4.2) with new text from Mark Jones
    5. Replaced section 5.1 with new text from Marc Hadley
    6. Merged XMLP_UnitData and XMLP_Intermediary into a single operation.
- Changes from Draft of 30th March 2001
    1. Removed issue 4 from frontpiece. resolved by inclusion of Fig 2.2 (later renumbered Fig 2.1)
    2. Removed references to XMLP_INTERMEDIARY operation. Was previously merged with XMLP_UNITDATA
    3. Fixed various typo's reported by Jean-Jacques Moreau
    4. Handed over editing to Mark Jones.
    5. Revised section 4 to reflect terminology agreement from April 4, 2001 telcon on with respect to handler and module.
    6. Revised section 4.2 to make block properties more abstract.
    7. Added Editors at beginning of document.
    8. Revised section 2 to introduce the simpler figure (now Fig 2.1) first and then then more complex figure (now Fig 2.2)

complex figure (now Fig 2.2).

- Changes from Draft of 17th April 2001
  1. Removed Issues list.
  2. Fixed various typo's.
  3. Changed wording at end of 5.1.4.1 to "HTTP persistent connections".
  4. Changed boilerplate to reflect linkage to XMLP/SOAP spec.
- Changes from Draft of 23rd April 2001
  1. Fixed Typographic errors reported by Gerd Hoelzing
  2. Updated Document Status.
  3. Added Acknowledgements ahead of Change Log