EG 2032-4-2007

SMPTE ENGINEERING GUIDELINE

Media Dispatch Protocol (MDP) — Engineering Guideline (Informative)

Table of Contents

Approved
December 14, 2007

Foreword

SMPTE (the Society of Motion Picture and Television Engineers) is an internationally-recognized standards developing organization. Headquartered and incorporated in the United States of America, SMPTE has members in over 80 countries on six continents. SMPTE's Engineering Documents, including Standards, Recommended Practices and Engineering Guidelines, are prepared by SMPTE's Technology Committees. Participation in these Committees is open to all with a bona fide interest in their work. SMPTE cooperates closely with other standards-developing organizations, including ISO, IEC and ITU.

SMPTE Engineering Documents are drafted in accordance with the rules given in Part XIII of its Administrative Practices.

SMPTE Engineering Guideline EG 2032-4 was prepared by Technology Committee S22 on Committee on Television Systems Technology.

Introduction

The Media Dispatch Protocol (MDP) is a means of orchestrating the delivery of media files over IP networks. It defines a standard mechanism for implementations to initiate a delivery, to negotiate the details of the delivery, to provide information about the progress of the delivery, and to provide a confirmation of the outcome of the delivery.

It is important to note that MDP is not a transfer protocol. Instead it allows organizations to transfer files at agreed times, using agreed transfer protocols, and using an agreed set of secure technologies (where appropriate). Furthermore, MDP is not overly prescriptive in regards to which protocols or technologies shall be used; rather it provides a framework which allows organizations to choose those that best suit their own needs.

This document is the MDP Engineering Guideline. It provides an introduction to the protocol, explains what problem it is intended to solve, and outlines how it works. It also provides information what is contained in the normative MDP specification documents.

1 Scope

This Engineering Guideline gives an introduction to the Media Dispatch Protocol (MDP) for the orchestration of the delivery of media files over IP networks. It describes the purpose of the protocol, and presents an outline of the technology involved, including an overview of the messages and data structures of the protocol, and how the protocol can be extended to meet future scenarios.

The Guideline provides advice on how the protocol can be used in practice, and what should be implemented. It also outlines how common transfer protocols and security mechanisms might be used in conjunction with MDP.

2 MDP Document Structure

The MDP specification is split into a number of separate parts (see Figure 1) in order to create a document structure that allows new applications to be covered in the future. These parts are:

Part 1 (Normative), the MDP protocol specification (SMPTE 2032-1).

Part 2 (Normative), the MDP mapping specifications (e.g. SMPTE 2032-2 is the MDP/XML/HTTP mapping).

Part 3 (Normative), the MDP profile specifications (e.g. SMPTE 2032-3 is the MDP Basic Target Pull profile).

Part 4 (Informative), the MDP Engineering Guideline (this document).

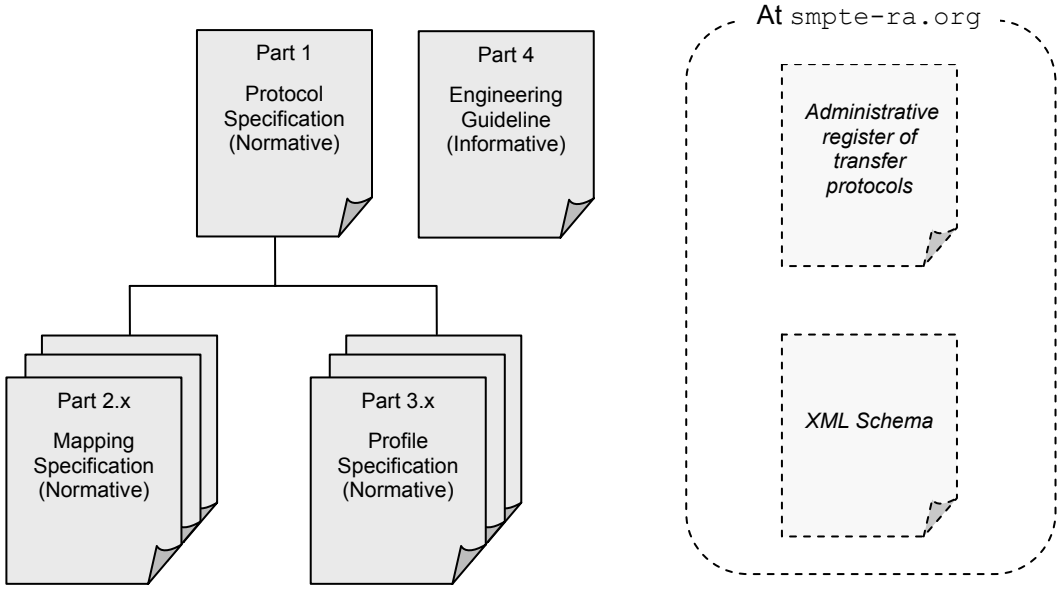

Figure 1 – MDP document suite

When implementing MDP, you should ensure that you have the latest version of all of these documents. The individual mapping and profile specifications will be independently updated.

This document is the MDP Engineering Guideline (Part 4). It provides an introduction and description. This document should be read first because it introduces many of the concepts and explains what problem MDP is intended to solve. It is presented as a set of questions (or an extended FAQ). Concepts are introduced gradually, and, where necessary, more detail is added later. This makes the document easier to read, but less good as a reference, so you are encouraged to make full use of the table of contents. Annex B of this document lists the defined terms in the specification.

The MDP protocol specification (Part 1) is at the core of the MDP standard. It defines the syntax and semantics of the messages and data structures used within the protocol.

The MDP mapping specifications (Part 2) define how the messages and data structures of Part 1 are represented on the network. At present there is only one defined mapping (MDP/XML/HTTP); however, as new user and technical requirements are identified, additional mappings will be produced. Each mapping specification will have its own document.

The MDP profile specifications (Part 3) define subsets of the messages and data structures of Part 1, and place constraints on the use of these subsets. The purpose of this is to make it easier to achieve interoperability between implementations. At present there is only one defined profile (Basic Target Pull); however, as new user and technical requirements are identified, additional profiles will be produced. Each profile specification will have its own document.

The specification also normatively references an “Administrative Register” document on the SMPTE Registration Authority website. This contains a list of transfer protocols that have defined names with MDP (for instance “HTTP”). This will be updated as required to incorporate new protocols without the need to issue a new version of a specification document.

The MDP document suite makes reference to several non-SMPTE standards. Unlike many SMPTE specifications, the majority of these are from the Internet and Web community. Many are “Request For Comment” documents (RFCs) from the Internet Engineering Task Force (IETF)”, or from the World Wide Web Consortium (W3C). A Bibliography for the document suite appears in the protocol specification.

To improve the clarity in all parts of the MDP document suite, `text in this typeface` is used to indicate the names or values of messages and data structures used in the protocol.

3 What Problem Does MDP Address?

The use of data files to represent audio and video content has revolutionized the television industry. For example, almost all editing occurs using non-linear systems, and most playout occurs from disk-based servers. However, at the time of writing (early 2007) transfer of content as files has so far largely been confined to scenarios where the content formats and transfer protocols can be carefully controlled. In particular, delivery between different organizations, or different departments of the same organization, still often occurs through the use of physical media such as videotapes, or via SDI and video lines.

As the cost of metropolitan- and wide-area connectivity falls, we can expect the demand to increase for file-based media delivery over IP-based networks. Examples include delivery of content between production facilities and broadcasters, and provision of agency news feeds. The success of such an approach depends on the availability of fit-for-purpose network interconnectivity, but it also depends on the systems at either end understanding one another. While the simple transfer of a file is a trivial problem (FTP is available on almost every platform), this only provides part of what is required. To avoid the need for manual intervention during every delivery, we need a mechanism to automatically initiate, supervise and audit the transfers. In addition because there are so many different transfer protocols in existence, and so many ways of configuring a transfer, the systems need to agree just what is going to happen.

There are on the market various proprietary file delivery solutions that solve these problems. However, differing systems do not interoperate, which in practice means that an organization wishing to exchange

content with multiple partners needs to deploy multiple systems. In many cases this would be unacceptable, and a standardized approach is needed.

MDP provides part of such a standardized approach. In particular it provides a mechanism to allow organizations to agree on the details of a delivery (what files are to be sent, when transfer will occur, what transfer protocol will be used), to manage its lifecycle, and to exchange information about its progress.

MDP was initially developed by a working group of the Professional MPEG Forum that investigated requirements and technologies for the exchange of large media files over IP networks. File delivery within the post-production community was seen as an important use case, and Annex A provides a summary of the relevant requirements identified by the group in this area.

MDP was therefore initially developed to be suitable for file transfer within the post-production community. However, it contains no features that tie it to such applications, and is suitable for use in other scenarios, for instance delivery of finished programs to playout centers. In fact, MDP is not specific to media files and could be used in any scenario requiring the transfer or large files between organizations.

4 How Does MDP Relate to…?

As MDP is in some ways an “invisible technology” it is helpful to clarify how it relates to other technologies.

4.1 Applications

A production or other media organization typically creates, manages and processes files using a selection of applications such as non-linear editing systems, media asset management systems and transcoding engines.

Sometimes these applications will need to exchange these files with applications belonging to other organizations. Because there are a wide range of possible mechanisms for performing these exchanges, it makes sense if these are not implemented in the applications themselves, and instead the tasks are delegated to specialized software agents. Then, as new transfer protocols, or other technologies, are adopted, it is not necessary to change the code within the applications themselves. As discussed in section 5, MDP provides a standard way for such agents to communicate.

4.2 Transfer Protocols

MDP is not a transfer protocol. Instead it carries information about what transfer protocol might be, or is being used. To ensure interoperability the standard requires all implementations to support HTTP and HTTPS, but allows the use of other protocols.

Although MDP was developed for deliveries over metropolitan- and wide-area networks, it can also be used with protocols more suitable for local-area networks, for example copying of files from a shared network file system such as NFS.

4.3 File Formats

Although MDP provides a logical grouping of files that are involved in the same delivery, it is not a file wrapper or container format like MXF (SMPTE 377M). It is expected that delivery of MXF files will be an important use of MDP, but the protocol makes no assumptions or requirements of the type of files transferred (they do not even have to be related to audio or video).

4.4 Security Technologies

Often it will be necessary to deliver files over public networks such as the Internet. MDP does not in itself ensure that deliveries are secured, but allows security technologies such as TLS (Transport Layer Security) to

be used, and enables implementations to exchange information about what security measures are supported before starting the transfers.

4.5 Metadata

MDP is not a metadata protocol, and it has no defined relationship with any metadata standard or recommended practice (e.g. SMPTE RP 210).

MDP can be used to transfer files containing metadata, e.g. within an MXF file or in a separate XML file.

The standard forbids the carriage of “dark metadata” within the protocol; this is discussed further in section 9.

4.6 Web Services

“Web service” simply means any software system that supports interoperable machine-to-machine interaction over a network. Therefore MDP implementations can be considered as web services.

However, “web service” is also often used to mean a service that uses SOAP (Standard Object Access Protocol) envelopes over the network and WSDL (Web Services Description Language) to describe its interface. At the time of writing, MDP does not use SOAP or WSDL; however, this could be added in a future mapping specification.

4.7 Middleware

“Middleware” can mean various things, but in general is software that connects other items of software. This could be for reasons of interoperability, or to provide a distributed architecture. MDP arguably could be seen as a type of middleware.

4.8 Proprietary Delivery Solutions

Many file delivery solutions currently on the market make use of high-performance, but often proprietary, transfer protocols. MDP is not intended to replace such solutions; the adoption of an MDP interface could aid the uptake of these new protocols, and is highly encouraged.

4.9 BXF

The Broadcast eXchange Format provides a standard for interchange of data and metadata among professional broadcast systems. It is intended for transferring schedule and playlist information between traffic, automation and similar systems. Although BXF includes messages to order content transfers, the intended application domain is different to that of MDP.

4.10 TV-Anytime, ADI, SIP/SDP, etc.

There are various other non-SMPTE protocols and interfaces that also cover some aspects of orchestration of content delivery, for example:

TV-Anytime defines standards for distribution of content for home storage.

ADI (Asset Distribution Interface) defines how to place content assets in context for video-on-demand distribution.

SIP (Session Initiation Protocol) and SDP (Session Description Protocol) manage and describe multimedia communication sessions, and the IETF is studying extension to support file transfer.

However, these are aimed at solving different problems to MDP, and are aimed at different application domains.

5 How Does MDP Work?

This section explains the main concepts within the protocol, with particular emphasis on the MDP/XML/HTTP mapping and the Basic Target Pull profile.

Figure 2 shows the context for which MDP is intended to be used, and what is defined in the protocol specification.

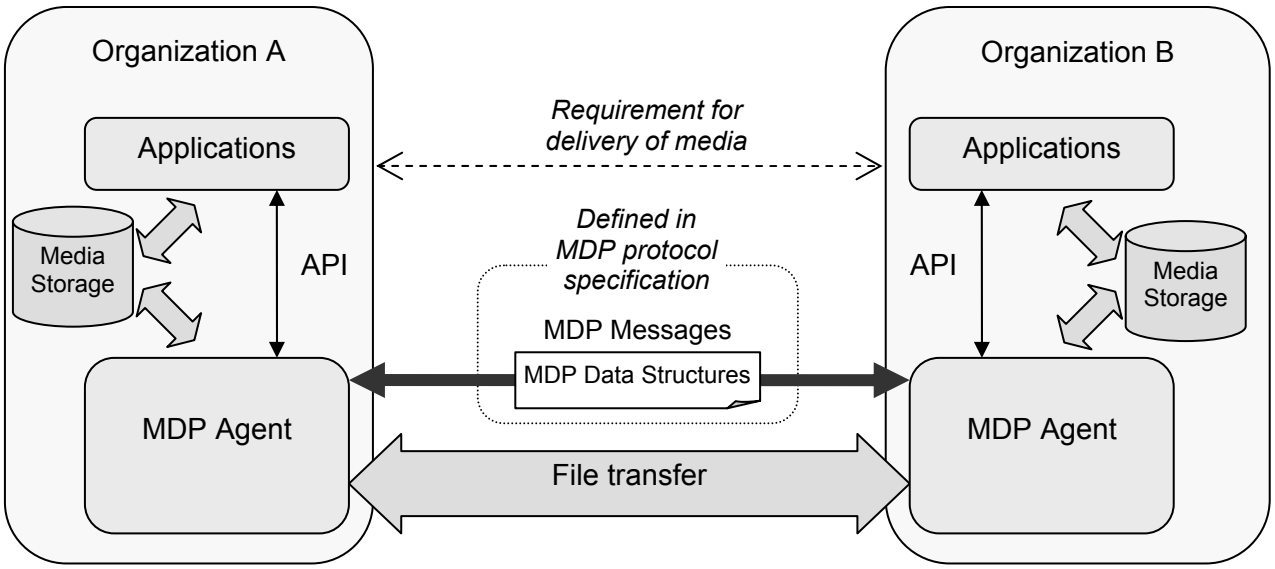

Figure 2 – Context for Media Dispatch Protocol

5.1 Organizations and Agents

The MDP specification uses the terms “organization” and “agent” extensively. The following summarizes their meaning:

> An agent manages deliveries on behalf of an organization.

An organization is someone or something that has a requirement for file delivery; this might be a private company, a public corporation, a department, possibly even an individual person.

An organization is identified within MDP by a simple string. A domain name such as `broadcaster.com` has the advantage of global uniqueness, but the standard does not require the use of these, and other identifiers could be used, for instance based on X.500/LDAP distinguished names (e.g. “`CN=Broadcaster`”).

An agent will normally be a piece of software, typically running on a computer with network connectivity. Generally an agent will offer some form of interface. This might be a user interface such as a web page allowing a user to request and check on a delivery. More importantly it might be API allowing an application such as an editing system or media asset management system to automatically invoke a delivery, and receive information about its progress. Please note that the MDP specification does not define or require any interface; this is out of scope of the standard.

An agent is identified by its message endpoint URL, as discussed in section 5.6.

Note: In practice, a single instance of agent software might handle deliveries on behalf of more than one organization (e.g. `mdpagent.com` might handle deliveries for `aaa.com`, `bbb.com` and `ccc.com`). However, this is of no relevance to the MDP standard, even if the agent uses the same message endpoint URL when acting on behalf of more than one organization.

5.2 Projects

Whatever the organizations are, there will always be a reason for deliveries between them, for example the deliveries might be regarding program number `P12345678`. In MDP this reason is called a "project", and is identified by a simple string. This string does not have to be globally unique; it just has to be sufficient to identify the project for the two organizations.

5.3 Transactions

The sequence of steps that the agents take with regards to a delivery (of any number of files) is called a "transaction". This starts with one agent telling another that a delivery is required. The agents then negotiate the details of how the delivery will occur (for example which files will be sent, what transfer protocol will be used, when transfers will start), and initiate file transfer operations as appropriate. During transfer, the agents can exchange information about the progress of each transfer. After the transfers have completed, the agents exchange details about the outcome of each transfer. See section 6 for more detail about the lifecycle of a transaction.

A transaction is identified within MDP by a simple string. This does not have to be globally unique; it just has to be sufficient to identify the transaction for the organizations and project involved. However, use of a globally unique identifier such as a GUID is suggested as a straightforward means of ensuring this is the case.

5.4 Initiator Target Agents

The specification uses the terms "initiator agent" and "target agent" extensively:

> A transaction starts with the initiator agent telling the target agent that a delivery is required.

5.5 Push-Mode, Pull-Mode and Use Cases

In the above statement, "A delivery is required" could mean several things. For the Basic Target Pull profile, the statement could be written as:

> A transaction starts with the initiator saying that it would like the target agent to pull some files across the network.

In this profile the initiator agent is typically "at the sending end". The target agent is "at the receiving end", and initiates all the transfers, in other words the transfers are pull-mode. MDP calls this a "target pull use case", hence the name of the profile. An example of a scenario that is well-suited to a target pull use case is where a production company delivers television programs to a broadcaster; the production company knows when the content is available and so is in the better position to initiate the transaction, while the broadcaster can initiate the transfers at times that are best suited to its own schedules.

Other use cases can be identified that may be better suited to other scenarios (see Figure 3):

> In an initiator push use case, the transaction is again initiated "at the sending end", but the initiator agent initiates push-mode transfers.
>
> In an initiator pull use case, the transaction is initiated "at the receiving end", and pull-mode transfers are used.

In a target push use case, the transaction is initiated “at the receiving end”, and push-mode transfers are used.

MDP can also (in theory) support mixed use cases, e.g. where the transaction is initiated “at the sending end” but some transfers are push-mode while others are pull-mode.

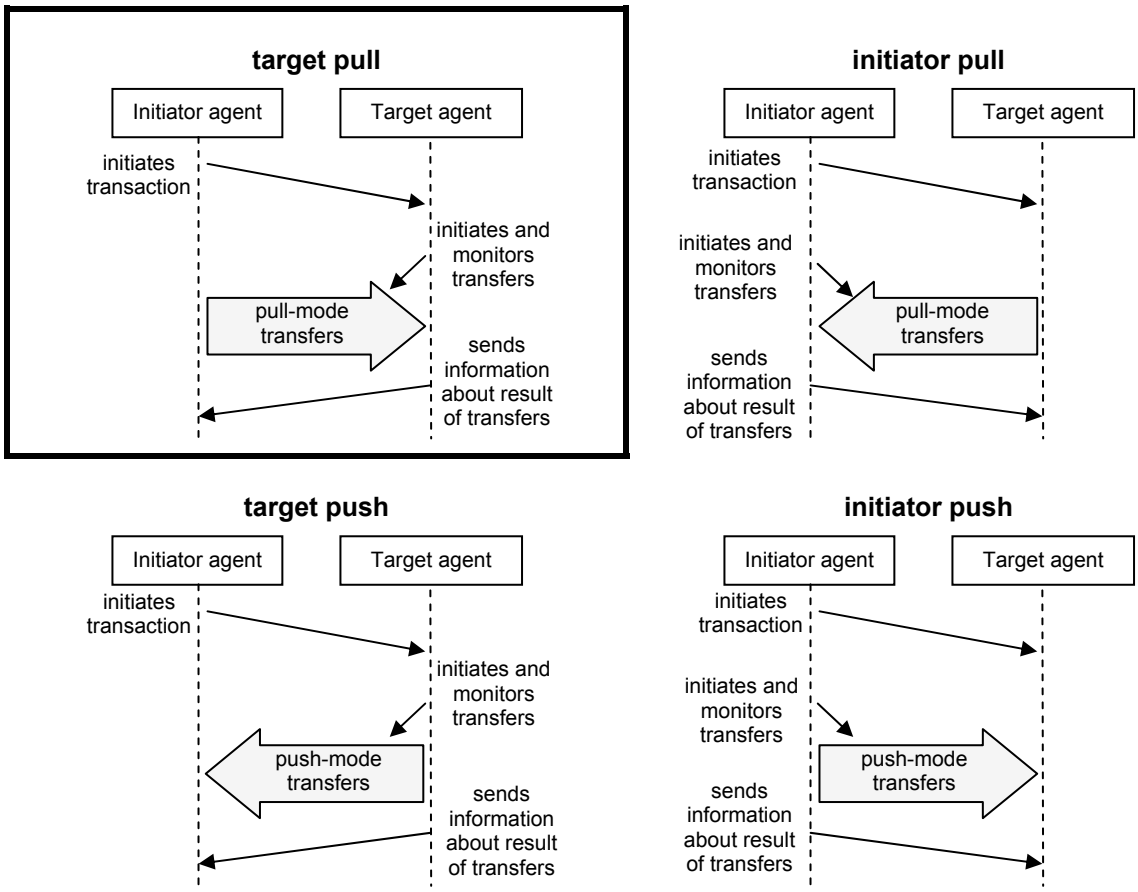

Figure 3 – Use cases

Although at the time of writing no profile exists that allows any use case other than target pull, the specification includes reserved values enabling these to be added in the future.

Note: For simplicity, transfers are shown between as occurring between the agents in Figure 3. As explained in section 5.10, this does not have to be the case. This is why the phrases “at the sending end” and “at the receiving end” are in quotation marks.

5.6 The Manifest Document (Part One)

The manifest document is fundamental to the way in which MDP works (in fact during development, the protocol was often just called “Manifest”). It is a document that is sent between the agents and provides information about the transaction. As the transaction progresses, the agents send one another manifest documents to negotiate the details of the transfers that will occur and provide information on their progress.
The information in the manifest document includes (among other things):

Identifiers for the organizations, agents, project and transaction.

A list of files that are offered for transfer, or that have been transferred.

Some information about the files (name, size, etc.).

How each file might be transferred, or is being transferred, including protocol, time scales, etc.

The status of each transfer (e.g. in progress, succeeded).

For the MDP/XML/HTTP mapping, the manifest document is an XML document whose root element is `<manifest…/>`.

The manifest document is covered in more detail in section 5.9, and how it is updated during a transaction is described in section 6.

Note: The MDP standard is only concerned with the use of the manifest document when it is sent over the network between agents. Implementers might choose also to maintain manifest documents internally within an agent, as a convenient means of storing the state of the transaction, but this is entirely outside the scope of the standard.

5.7 Messages and Endpoints

As shown in Figure 2, agents communicate with each other using MDP messages. These have meanings such as:

I wish to initiate a new delivery, and here is a manifest document with the details.

Here is a manifest document with what I agree to for this delivery.

Please tell me how the delivery is progressing.

Here is a manifest document with the details of how the delivery is progressing.

Please pause these file transfers.

I wish to terminate the delivery.

The delivery has completed, and here is a manifest document with the details of what happened.

To comply with the standard, an implementation must be able to both send and receive messages, even if it only acts as an initiator agent or only as a target agent.

MDP defines four types of message: commands, confirmations, queries and replies.

Commands and confirmations are used together: one agent sends a command to make something happen, and the other agent responds with a confirmation containing either an acknowledgement or an indication of an error.

Similarly, queries and replies are used together: one agent sends a query to request some information, and the other agent responds with a reply containing either the requested information or an indication of an error.

Commands and queries take parameters containing information relevant to the command. Examples of parameters include project identifiers and manifest documents. Different commands and queries require different combinations of parameters, and some parameters are optional.

In the MDP/XML/HTTP mapping, detailed in SMPTE 2032-2, messages are implemented using an approach that is commonly adopted for submitting information to a web site. One agent sends an HTTP POST request

containing the message parameters and the other agent sends back an HTTP OK response containing the acknowledgement, requested information, or error.

Note: Using the terminology of the HTTP/1.1 specification, information is included in an HTTP POST request as "parameters". SMPTE 2032-2 is normative so must use the same language, so when reading it, be careful not to confuse parameters of the MDP message with the parameters of the HTTP request; the latter is specific to the MDP/XML/HTTP mapping.

An agent implementation must make a message endpoint available. This is a URL associated with the agent to which other agents can send messages. An example of an agent URL that might be used with the MDP/XML/HTTP mapping is `https://broadcaster.com/mdp`. The specification assumes that the initiator agent already knows the message endpoint of the target agent; this is discussed further in section11.

The protocol specification (SMPTE 2032-1) includes a full description of MDP's set of messages and parameters, and the MDP/XML/HTTP mapping specification gives examples of HTTP requests and responses for some messages.

5.8 Data Structures

Some of the information sent within messages have a simple type, for instance the `txorg` parameter is a simple string. Other information has a more complex type, defined in the data structures part of the protocol specification (SMPTE 2032-1). When using the MDP/XML/HTTP mapping, these structures are sent as XML elements within the message.

MDP defines three types of data structure:

Elements that contain other structures

Lists of other data structures (e.g. `comment_list`)

Properties that take simple values (e.g. `transactionid`)

The protocol specification (SMPTE 2032-1) specifies what data structures are allowed in MDP messages.

The MDP/XML/HTTP mapping specification (SMPTE 2032-2) references an XML schema document located on the SMPTE Registration Authority website. This specifies the XML representation of the data structures. It is recommended that implementations validate any received XML against this schema.

Some properties must be set to a value taken from a list of allowed values. These are detailed in the data dictionary section of the protocol specification (SMPTE 2032-1). An example of such a property is `profile`, which must be set to a value corresponding to an MDP profile specification, for example "`basic target pull`".

For some properties, the specification also allows agents to use values that do not appear in the list of allowed values, provided that the value starts with "`X-`" or "`x-`". This allows "non-standard" transfer protocols to be used in a transaction; see section 8.3 for more details.

5.9 The Manifest Document (Part Two)

We can now look at the manifest document in more detail. Figure 4 shows the higher-level lists and elements present in the manifest document, and their more important properties. The relationship between these can also be described in words as follows:

A manifest element contains lists of `transferoption` and `file` elements. Each `file` element contains a list of `transferwith` elements, each of which references one of the `transferoption` elements.

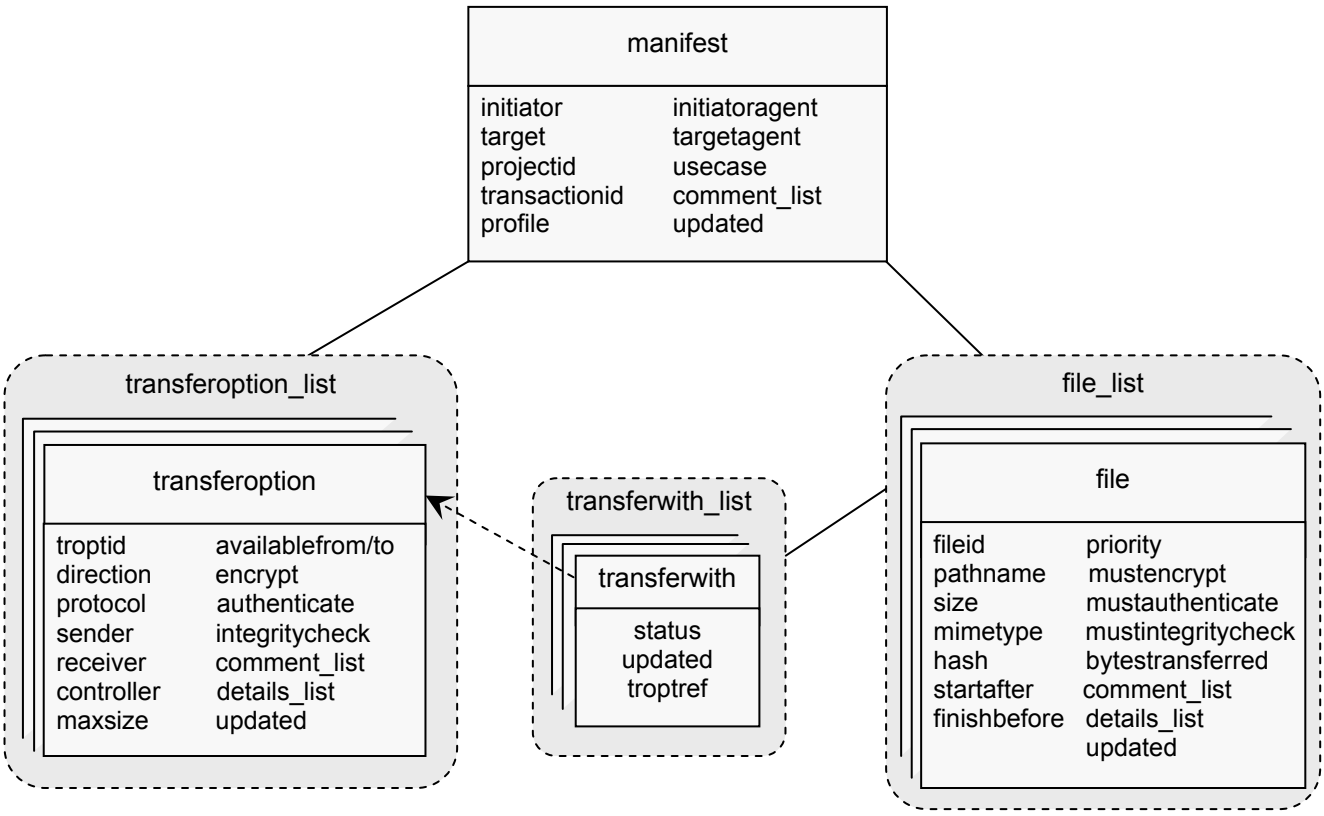

Figure 4 – Summary of manifest document data structures

Annex A of the protocol specification (SMPTE 2032-1) gives an example of a manifest document that complies with the Basic Target Pull profile.

5.9.1 `manifest` elements

The properties of a `manifest` element include identifiers for the transaction and for the project, organizations and agents involved in the transaction. Another property indicates which profile the transaction conforms to; this allows the target agent to determine easily whether it can support the transaction.

5.9.2 `transferoption` elements

Each `transferoption` element in the manifest document represents one possible method for performing file transfer. The properties of a `transferoption` element provide information about whether the method is pull-mode or push-mode, the transfer protocol used (e.g. HTTP), and how large a file can be transferred. The `sender` and `receiver` properties provide the base URL where files can be pulled from or pushed to respectively. There are also optional properties describing how transfers are secured, and when transfers can occur.

The agents use the `transferoption` elements as part of the “negotiation” phase of a transaction (see section 6.3). In a Basic Target Pull profile transaction, the initiator agent provides all the possible `transferoption` elements in the initial manifest document, and the target agent selects which to use.

5.9.3 `file` elements

Each `file` element in the manifest document represents one file that might be, is being, or has been transferred. The properties of a file element provide some basic information about the file itself, including its name, size, MIME media type, and optionally a hash or digest value for checking the integrity of the transfer.

The properties may also specify requirements on the transfer, including when it must occur, what its relative priority is, and how it must be secured. Finally the `bytestransferred` property is updated during transfer to indicate how much of the file has been transferred so far.

The `name` property can include "`/`" characters; this allows files to be transferred from nested directories or folders. More on this in section 5.10.

5.9.4 `transferwith` elements and `status` properties

The list of `transferwith` elements within a `file` element indicates which transfer options relate to the file: each `transferwith` element includes a property (`troptref`) that references a `transferoption` element.

A `transferwith` element therefore corresponds to a single potential or actual file transfer. Typically each file will have a `transferwith` element for each `transferoption` element in the manifest document. However, this need not be the case. In the example partial manifest document below, the four files to be sent reside on two different servers, and so there are two `transferoption` elements, each with a different `sender` property, and each `file` property has a `transferwith` corresponding to the appropriate `sender`.

```
<manifest>
       <profile>basic target pull</profile>
    ...
    <transferoption list>
       <transferoption>
              <troptid>HTTP GET from server1</troptid>
            <protocol>HTTP</protocol>
              <direction>PULL</direction>
              <sender>http://server1.org.com/files</sender> ...
       <transferoption>
       <transferoption>
              <troptid>HTTP GET from server1</troptid> ...
              <sender>http://server2.org.com/files</sender> ...
       <transferoption>
    </transferoption list>
    <file list>
       <file> ...
              <pathname>prog1.mxf</pathname> ...
              <transferwith_list>
                     <transferwith>
                            <troptref>HTTP GET from server1</troptref> ...
                     </transferwith>
       </file>
       <file> ...
              <pathname>prog2.mxf</pathname> ...
              ...
                            <troptref>HTTP GET from server1</troptref> ...
              ...
       </file>
       <file> ...
              <pathname>prog3.mxf</pathname> ...
              ...
                            <troptref>HTTP GET from server2</troptref> ...
              ...
       </file>
       <file> ...
              <pathname>prog4.mxf</pathname> ...
              ...
                            <troptref>HTTP GET from server2</troptref> ...
              ...
       </file>
    </file_list>
```

The `transferwith` element's `status` property indicates whether the transfer:

has been offered by one of the agents (the initiator agent for Basic Target Pull profile)

was rejected by the other agent

was accepted but has not started yet

is in progress

has been intentionally paused

has stalled (become too slow to be useful)

has succeeded

has failed

The protocol specification (SMPTE 2032-1) specifies how the value of this property is allowed to change as the transaction progresses.

5.9.5 `details`, `updated` and `comment` properties

The manifest document also includes properties to allow:

links to external metadata and other information (see section 9)

audit information about which agent created or updated each major element and when

human-readable comments for debugging or administration purposes

5.10 Transfers, Controllers, Senders, and Receivers

As well as sending each other messages, agents also are responsible for initiating the actual file transfers. The `controller` properties in the manifest document indicate which organization's agent has the responsibility for each transfer. In the case of the Basic Target Pull profile, `controller` is always the target agent.

Each file transfer has a "transfer endpoint"; the URL from/to which the controller agent pulls/pushes the file. (Do not confuse these with message endpoints.)

For a pull-mode transfer, the controller agent determines the transfer endpoint for each file by concatenating the relevant `sender` and `pathname` properties in the manifest document (with a "/" character in between if required). For example:

```
sender = https://mdp.facility.com/transfer/P0123456

pathname = prime/program.mxf
```

Controller gets file from `https://mdp.facility.com/transfer/P0123456/prime/program.mxf`

For a push-mode transfer the `receiver` and `pathname` properties are similarly concatenated.

MDP does not specify any location at the controller's end of the transfer. For a pull-mode transfer the controller can store the received files at any location, and for a push-mode transfer it can take them from any

location. In addition, MDP does not require the controller to preserve any directory structure implied by "/" characters in the value of `pathname`. However, it may be of benefit to do so in many cases (so in the example above, the controller would create a "`prime`" directory and put `program.mxf` into it).

Remember those quotes around "the sending end" earlier? As shown in Figure 5, the transfer endpoint from which the target agent pulls files might be located on a different host to the initiator agent, possibly part of a different domain, and maybe even owned by a different organization. This is supported in MDP and in the Basic Target Pull profile, and simply means that the `sender` and `initiatoragent` properties in the manifest document refer to different hosts. Similarly, for push-mode transfers, the receiver transfer endpoint need not be on the same host as the agent at "the receiving end".

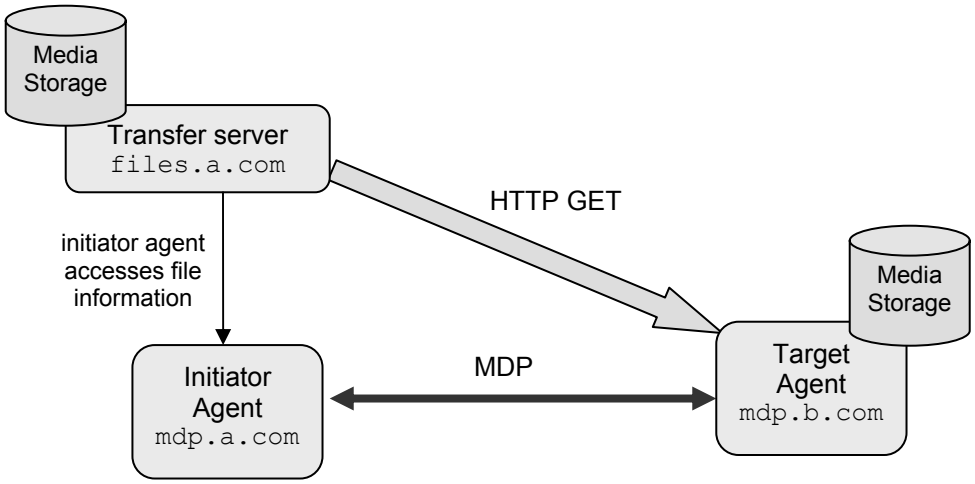

Figure 5 – Separate MDP agent and transfer server

Note: Although Figure 5 shows the target agent performing the HTTP GET, in some implementations the agent might delegate this to a separate host. This does not affect the use of MDP itself.

5.11 The Capabilities Document

Support for some parts of MDP is optional. In addition agent implementations might place limits on what can be supported, an example being the maximum number of files in a transaction. An agent can request that another agent send a document, known as the capabilities document, describing the features that it supports, and associated limits. By requesting a capabilities document first, an agent can avoid initiating a transaction that will clearly not be supported by the other agent.

The capabilities document contains a `capabilities` element, whose properties include:

- which profiles and mappings the agent supports
- which transfer protocols the agent supports
- how large a manifest document can be (total size, number of files, etc.)
- maximum size of file that can be transferred
- how transfers can be secured

Although most properties in the capabilities document are optional, it is recommended that implementations provide all information possible.

6 What Happens During a Typical Delivery?

This section describes the actions that occur in a successful delivery, with a particular emphasis on the MDP/XML/HTTP mapping and the Basic Target Pull profile. The diagrams in this section show examples of the messages exchanged between the agents, together with relevant sections of the XML manifest and capabilities documents. Further informative examples are presented in Annex A of the MDP specification (SMPTE 2032-1).

Note: To aid readability of the XML examples, list elements (those ending in "`_list`") are not shown if they only contain one element.

6.1 Before Initiation

Although not required by the standard, it is recommended that the initiator agent first ask for the capabilities document of the target agent by sending a `qry_requestcapabilities` query, especially if this is the first delivery between the agents for some time. This avoids the unnecessary sending of a (potentially large) manifest document to a target agent that does not support the transaction, or is simply not responding. The target agent sends its capabilities document back using an `rpl_capabilities` reply.

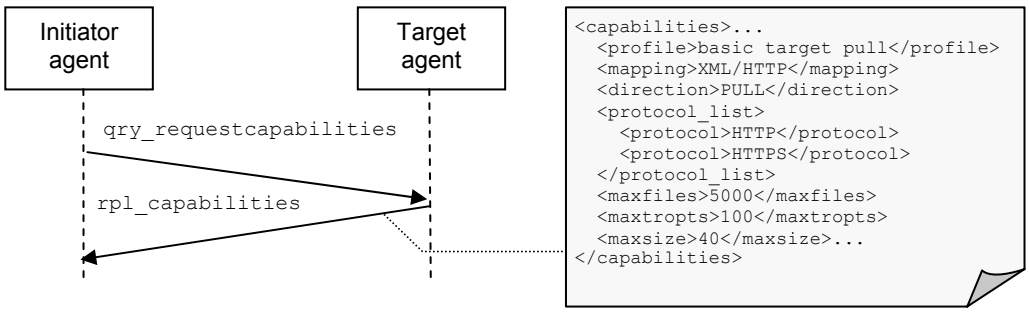

Figure 6 – Initiator agent requests capabilities document

6.2 Initiation

The initiator agent sends a `cmd_initiatingtransaction` command to the target agent. This includes a `manifest` parameter with an initial manifest document for the transaction. The target agent confirms that it agrees to the transaction with a `cnf_ok` confirmation.

For the Basic Target Pull profile, the manifest document includes all the files that might be delivered, and all the transfer options that might be used. All `status` properties are set to `offered`.

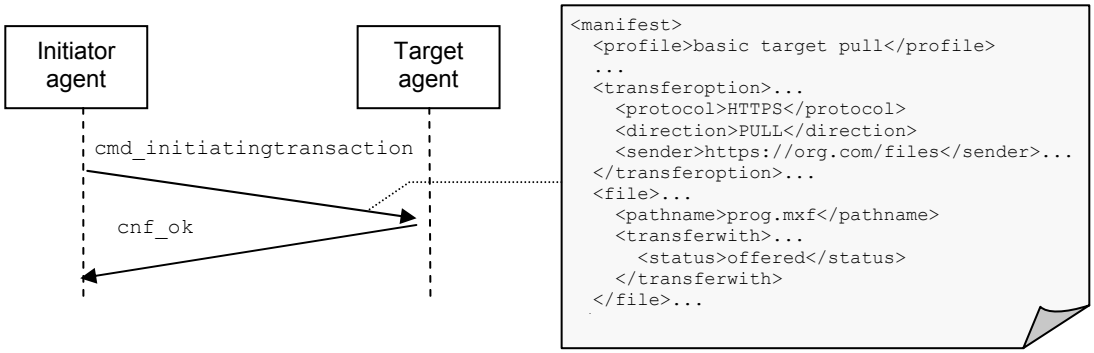

Figure 7 – Initiation

6.3 Negotiation

The agents then agree the details of the transfers. They do this by sending each other `cmd_sendingmanifest` commands, each with a new version of the manifest document. The `status` properties indicate which transfer options are accepted or rejected. The final manifest document during negotiation contains the information required to determine how the transfers should happen.

In general, negotiation could go on indefinitely, and agents could even add new files and transfer options as the negotiation processes. However, the Basic Target Pull profile simplifies things considerably: the target agent either accepts or rejects each transfer option in the manifest document, the initiator agent returns a `cnf_ok`, and the negotiation is completed.

Because there is no "accept/reject" property for a file in the manifest, an agent rejects a particular file outright by setting the `status` properties of all `transferwith` elements for that file to `rejected`.

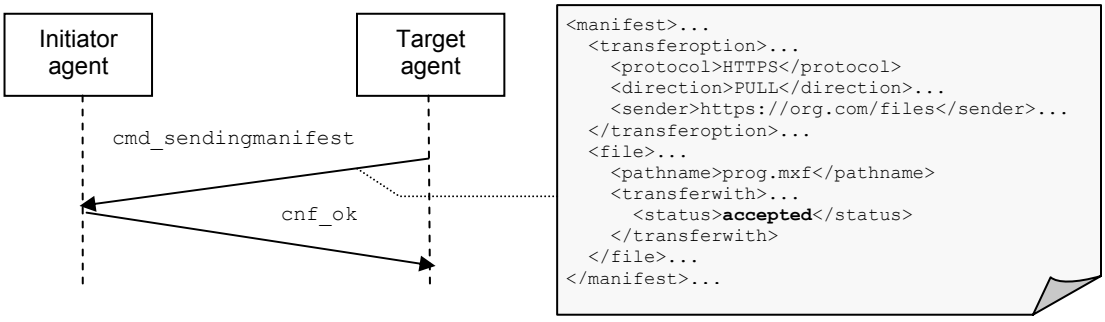

Figure 8 – Target agent accepts or rejects proposals

6.4 Start of Transfer

The `controller` properties in the manifest document indicate which agent is responsible for initiating each transfer. For the Basic Target Pull profile this is always the target agent. (In practice the target agent might delegate the task to some other transfer agent, but the standard is not concerned with such implementation details).

Section 5.10 explains how to derive the transfer endpoint; i.e., the URL from which the file is pulled.

Files can be transferred in any order, simultaneously or sequentially. The manifest document allows some scheduling and prioritization information to be exchanged between the agents; see section 11.

After at least one transfer has started, the target agent informs the initiator agent of the fact by sending a `cmd_sendingmanifest` command with the details, and the initiator agent acknowledges this with a `cnf_ok`.

Note: The protocol does not currently allow an agent to notify when each individual transfer starts. Instead, the other agent can "poll" using the status check facility described below. SMPTE 2032-1 includes reserved messages and data structures for a possible extension to allow asynchronous notification.

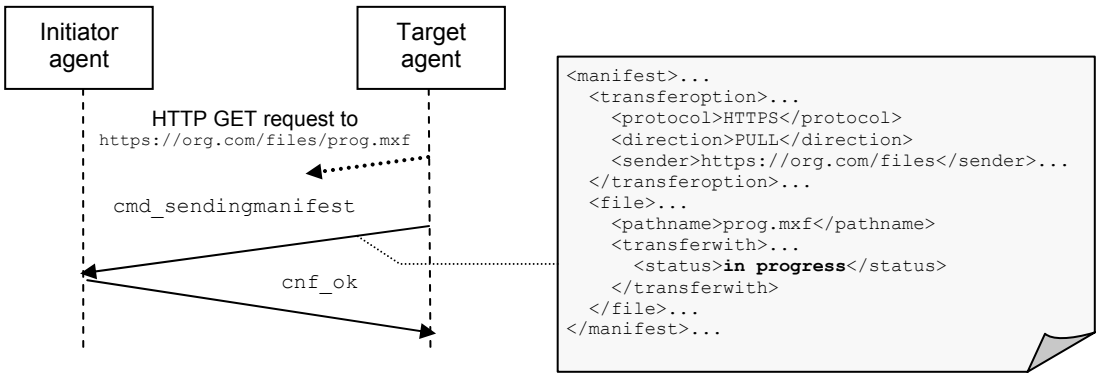

Figure 9 – Target agent initiates transfer

6.5 During Transfer

6.5.1 Status check

Agents can send `qry_requestm`anifest queries to receive a manifest document with information about how the transfers are progressing. (This can also be sent before any transfers have started.)

For the Basic Target Pull profile, only the initiator agent can send `qry_requestmanifest`. The target agent returns an `rpl_manifest` reply in which the `status` properties show the current state of each transfer, and the `bytestransferred` properties show how many bytes have been received.

In order to provide accurate information, it is recommended that the agent responsible for controlling the transfers regularly monitor their progress.

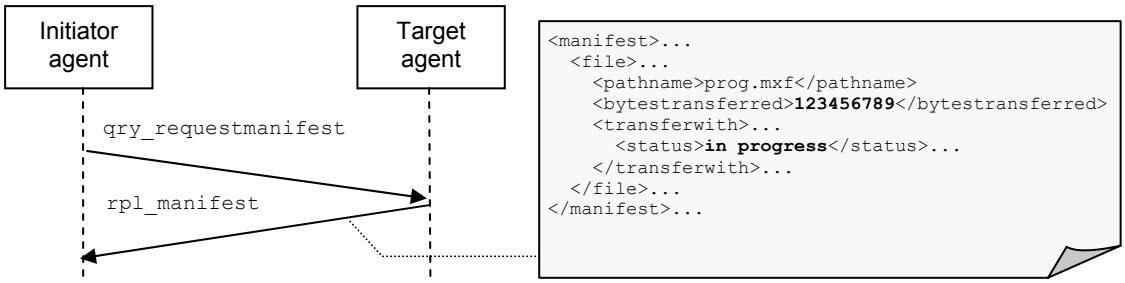

Figure 10 – Initiator agent requests progress check

6.5.2 Pause and resume

For many transfer scenarios, circumstances may change while transfers are in progress. For example a “rush job” may suddenly appear which needs to take priority over all other transfers. In such cases, rather than cancel the other transfers that are in progress, it is often better to pause (temporarily stop) them, and then resume them from where they left off, after the rush job has been completed. Another example might be when an operator has to intervene at the receiving end because a file server is about to become full.

If the requirement to pause the transfer(s) is at the controller end, then the controller agent can just go ahead and do so. However, often it will be at the non-controller end. MDP therefore allows the non-controller agent to request that one or more transfers be paused, and later resumed again, without any loss of data.

For the Basic Target Pull profile, the initiator agent sends a `cmd_pausetransfers` command, identifying the files to be paused, and the target agent acknowledges with a `cnf_ok`. However, this acknowledgement does not necessarily mean that the transfer(s) actually were paused, and it is recommended that the initiator agent send a `qry_requestmanifest` to find out what happened.

Similarly an agent sends a `cmd_resumetransfers` to request that one or more transfers are resumed from where they left off.

Some transfer protocols or common implementations of transfer agents (e.g. a widely-used SFTP implementation) do not support pause and resume, and the Basic Target Pull profile does not require agents to implement it. However, it is recommended that implementations do implement it at least for HTTP and HTTP/TLS transfers. Typically an agent implementation performing an HTTP GET transfer might accept a "chunk" of data (e.g. a megabyte), then check for any new MDP messages that have arrived; if a `cmd_pausetransfers` was not received it would then go on to the next chunk.

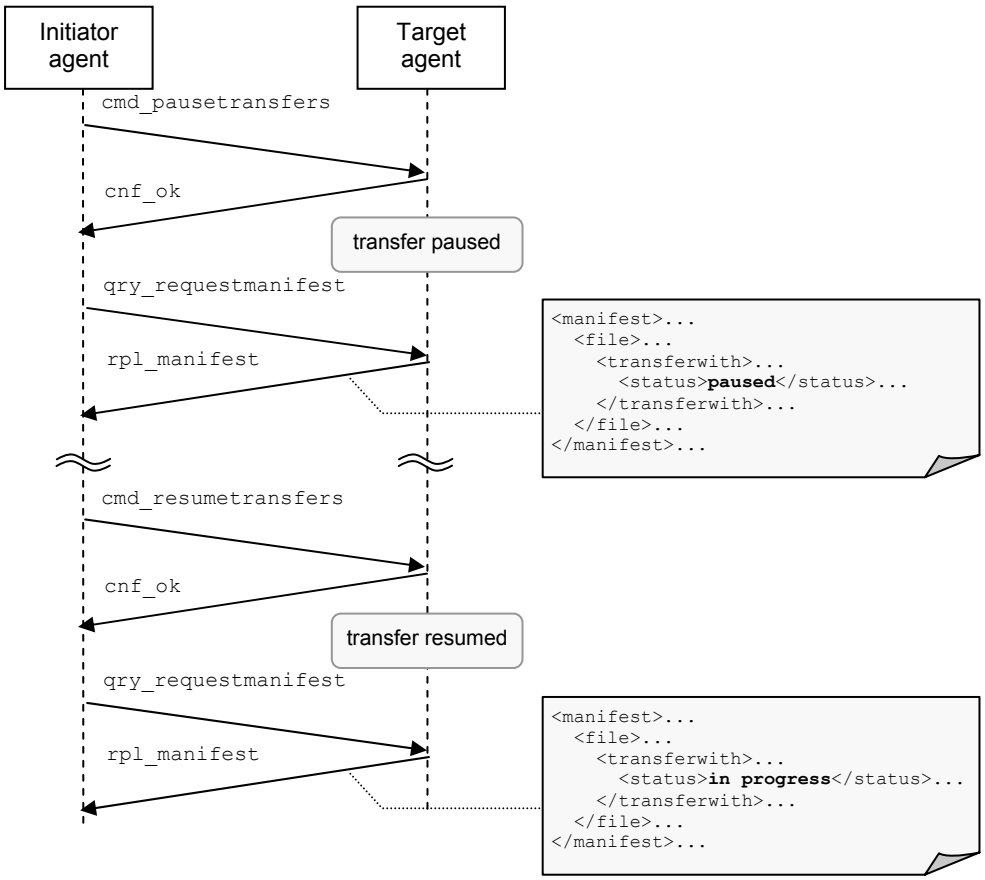

Figure 11 – Pause and resume

6.5.3 Checking received files

For the Basic Target Pull profile, the target agent checks that the name and size of each received file are as stated in the manifest document. It also may also calculate its hash (digest) value and check this also matches; see section 14.3.

6.6 Completion

Completion means that the transaction has finished. This may be because the negotiation failed to reach an agreement (all files were rejected). More typically, it means that all file transfers have succeeded or failed. For the Basic Target Pull protocol, the target agent sends a `cmd_completingtransaction` command, with a final manifest document showing the state of each transfer. The initiator agent acknowledges this with a `cnf_ok`, and the transaction is over.

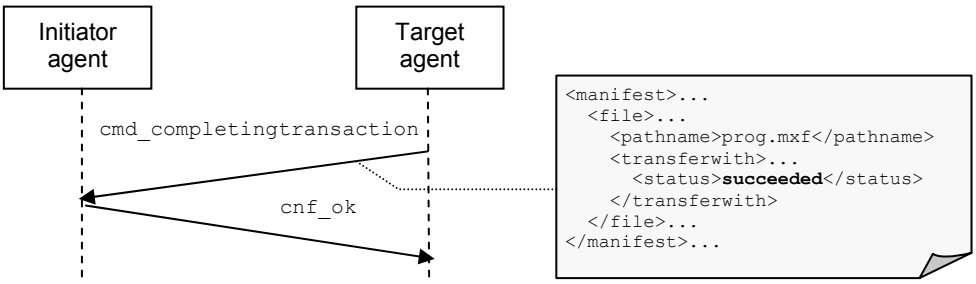

Figure 12 – Target sends notification of completed transaction

7 What Happens When Things Go Wrong?

7.1 General

The previous example showed a successful transaction. Of course, this will not always be the case. The following are some of the things that might go wrong:

An agent fails to receive a message.

An agent receives a syntactically incorrect or invalid message

The target agent does not support the profile required for the transaction.

The target agent does not agree to the transaction at all.

An agent does not support the functionality required (e.g. pause/resume).

No agreement is reached on the `files` and `transferoptions` (all `status` properties are set to `rejected`).

A transfer fails to start.

A transfer fails part-way through.

A transfer starts but becomes stalled (too slow to be useful).

A file that was received was not the correct file.

A file is corrupted during transfer.

With the exception of the first example, MDP provides a means for agents to inform each other of such problems.

7.2 Error Responses and `reason` Properties

To indicate a problem with a received command or query, an agent can respond with `cnf_error` or `rpl_error` respectively. Each of these contains an `error` data structure, which contains `reason` and `comment` properties.

In some places, the standard requires `reason` to be set to a particular value (e.g. if pause and resume are not supported for a particular transfer or transfers, `NOT_SUPPORTED` should be used). In general, however, the standard requires agents to use the most appropriate value, according to the descriptions in the definition of `reason`. The `comment` property allows implementations to include an additional human-readable explanation as to why the error occurred.

Here are some specific examples to clarify the use of `reason`, assuming the use of MDP/XML/HTTP mapping and Basic Target Pull profile:

A message with an unknown name (e.g. `cmd_requesttransfer`) is indicated using `INVALID_PARAM`.

A non-well-formed XML document is indicated using `BAD_PAYLOAD`.

An XML document that is invalid according to the MDP schema is indicated using `INVALID_PAYLOAD`.

A project identifier that is unknown to the agent receiving the message is indicated using `INVALID_ID`.

A manifest document that has fewer `file` elements than a previous one in the same transaction is indicated using `MANIFEST_ERROR`.

Because of the wide range of possible error conditions, the standard does not in general specify a particular agent behavior after a `cnf_error` or `rpl_error` has been received. However, it is recommended that if the negotiation has not yet been completed, the agent send a `cmd_abortingtransaction` command, and consider the transaction terminated. See also section 7.4.

7.3 Stalled and Failed Transfers and `updatereason` Properties

To indicate a problem with a transfer that has stalled or failed, an agent sends a manifest in which the relevant `transferwith` element has its `status` property set `stalled` or `failed`, and its `updatereason` property set to the most appropriate value defined in the specification. The `comment` property also allows implementations to include an additional human-readable explanation as to why a transfer failed.

Note: `failed` is used both in cases where the transfer fails to start, and where it starts and subsequently fails. The relevant `bytestransferred` property is set to zero if it does not start.

In the Basic Target Pull profile, the target agent only sends this information when requested to do so following a `qry_requestmanifest` query from the initiator agent. (Although the target agent sends a `cmd_sendingmanifest` at start of transfers, it cannot send further notifications until all transfers have completed.)

Here are some specific examples to clarify the use of `updatereason` for failed HTTPS GET transfers:

An HTTP 404 error is indicated using `FILE_NOT_FOUND`.

An HTTP 500 error (internal server error) is indicated using `TRANSFER_DID_NOT_START` or `TRANSFER_PROCESS_DIED`, depending on when it occurred. The `comment` property can mention the 500 error.

An internal error that occurs on the receiver side (rather than at the HTTP server) is indicated by `INTERNAL_ERROR`.

A received file that was not the same size as declared in the manifest document is indicated using `INCORRECT_FILE_SIZE`.

A received file that fails its integrity check (see section 14.3) is signaled using `INCORRECT_HASH`.

If a transfer slows to a halt, or nearly to a halt, then a transfer might be considered as stalled if there is a chance that it might restart, otherwise it will be considered as failed, with `updatereason` set to `LINK_TIMED_OUT`. The MDP specification does not specify how slowly a transfer has to be progressing to be considered to have stalled, nor for how long it should remain so before becoming failed. Implementers can choose these parameters to suit the expected transfers and network architecture.

7.4 Problems that MDP Cannot Express

What about the first example in section 7.1: an agent fails to receive a message? The agent sending the message will not receive a `cnf_error` or `rpl_error`; typically the HTTP request will timeout instead (assuming MDP/XML/HTTP mapping). The timeout period will be implementation-dependent and is not discussed further here.

In such cases, the MDP specification does not require the agent sending the message to take any particular action. However, it is recommended that if the negotiation has not yet been completed, the agent send a `cmd_abortingtransaction` command, and consider the transaction terminated.

Whether a transaction should be terminated if there is a communication failure between agents after transfers have started is more difficult. In many cases, the transfers will also fail. However, in some cases the transfers may still progressing well, and abandoning them because of a temporary problem with an agent's HTTP server might be a mistake. On the other hand, if that problem turns out not to be temporary maybe it would not be a mistake. Such problems will not be discussed further here.

Examples of other problems that are out of the scope of MDP are:

A received file had the correct name and size (and passed any integrity check) but for some other reason was not what was expected, for example it was the wrong format. Note that this includes the case where the file received is not compatible with the corresponding `mimetype` property in the manifest document; this property is only a hint about the type of the file.

A received file was lost or corrupted after the transaction was completed. (If the problem occurred before completion, the appropriate `status` property can be set to `failed` with `reasoncode` set to `INTERNAL_ERROR`.)

8 How Will MDP Be Kept Both Interoperable and Future-Proof?

8.1 Additional Profiles

The MDP Basic Target Pull profile has been specified first, as this best met the user requirements that were identified during the development of the protocol. Alternative profiles may be better suited to other applications of MDP. For instance, some applications may prefer push-mode transfer, while others may require the transaction to be initiated at the receiving end. As required, new profile specifications will be included into Part 3 of the standard.

In anticipation of the relevant specifications, the data dictionary includes "`basic target push`", "`basic initiator pull`" and "`basic initiator push`" as reserved values for `profile`. Like the Basic Target Pull profile, these profiles would use a simple negotiation procedure, but would support each of the other three use cases shown in Figure 3.

The dictionary also reserves the `basic` value. The Basic profile would also use a simple negotiation procedure but would be more flexible in supported use case; e.g. it would allow any combination of `direction` and `controller` in the manifest document. Finally the `full` value is also reserved for a possible profile that allows a greater degree of flexibility in the transaction, particular the negotiation.

To aid interoperability, it is recommended that an implementation support the Basic Target Pull profile, even if it will mainly be used with other profiles. It is also recommended that the simplest profile be used that supports a particular transaction; for instance, if all the `transferoptions` are compatible with the Basic Target Pull profile, then this should be signaled in the manifest document, rather than the more complex Basic profile.

8.2 Additional Mappings

The MDP/XML/HTTP specification was developed to provide a straightforward mapping that can be implemented on a wide range of platforms. Alternative mappings may be better suited to other applications of MDP. As required, new mapping specifications will be included into Part 3 of the standard.

Of particular interest is how MDP might be integrated into web services based architecture. Here a mapping based on suitable web services technologies would be appropriate. For instance, the MDP messages could be carried in a SOAP (Simple Object Access Protocol) envelope. To allow for a future MDP/SOAP/HTTP mapping specification, the dictionary entry for `mapping` includes `SOAP/HTTP` as a reserved value.

To aid interoperability, it is recommended that an implementation support the MDP/XML/HTTP mapping, even if it will mainly be used with other mappings. At the very least it is helpful if it can send its capabilities document with MDP/XML/HTTP so that other agents can see what mappings are supported.

8.3 Transfer Protocols

The dictionary section of the protocol (in SMPTE 2032-1) specifies the following standard protocols:

HTTP/1.1

HTTP/1.1 over TLS (also known as HTTPS)

FTP

FTP/TLS (also known as FTPS)

SFTP (an FTP-like protocol within the SSH suite, and not related to FTPS)

Anonymous FTP

FLUTE (see section 19)

NFS (see section17)

SCP (a remote copy protocol within the SSH suite)

Agents can use any of these in an MDP-compliant transaction, subject to any profile-specific rules. For instance the Basic Target Pull profile (SMPTE 2032-3) requires that the protocol supports a pull-mode transfer.

To ensure interoperability, the Basic Target Pull profile also requires all implementations to support both HTTP and HTTPS (efficient HTTP(S) servers are now available for most platforms). It is recommended that the initiator agent find out what additional protocols are supported from the target agent's capabilities document, and choose the `transferoption` elements in the manifest document accordingly. If the target agent does not provide this agent, then the initiator should include either HTTP or HTTPS, according to whether a secure transfer is required.

However, there are many other protocols available that are suitable for transfer of large files, and some of these are also included in the dictionary. Many protocols are optimized for high bandwidth wide area networks where TCP-based protocols such as HTTP and FTP can perform poorly (see section 13). These include new "open" protocols that one day could become standards (GridFTP is such an example), as well as proprietary protocols. Some protocols are "better" versions of HTTP or FTP, while others are based upon UDP rather than TCP.

One of the requirements when designing MDP was that users should not be stopped from using a transfer protocol just because it does not appear in the paper specification. MDP therefore allows agents to specify another protocol by prefixing the value of the `protocol` property with "`X-`" or "`x-`", e.g. `<protocol>X-NEWFTP</protocol>`.

Using the "`X-`" approach is convenient, but could lead to interoperability problems (for example, two implementations might use `X-NEWFTP` and `x-NewFTP` to indicate the same protocol). To make it as easy as possible for new protocols to be named in a standard way, the `protocol` dictionary table is actually an "Administrative Register" rather than a static table. This means the SMPTE Registration Authority maintains an online document with the most up-to-date version of the table (this is the normative version; the version in the paper document is informative). This can be updated much more easily than issuing a new version of SMPTE 2032-1. It is important that you check this, rather than rely on the paper version, which may be out-of-date.

To aid automation, the online `protocol` register is specified in a machine-readable format. For the MDP/XML/HTTP mapping, this is XML, as defined in the mapping specification (SMPTE 2032-2). The MDP XML schema also includes the definition of the XML in the register. Other representations might also be provided in the future to support other mappings.

The URL of the XML online register document is given in the specification. Implementers interested in adding new protocols can apply to SMPTE-RA; this is a relatively simple procedure and is described in SMPTE 2032-1.

The transfer endpoint URLs derived from `sender` or `receiver` and `pathname` need to be appropriate for the protocol in use, for example:

```
https://www.organization.com/mdp/files-out/content.mxf

ftp://user:password@ftp.organization.com/mdp/files-out/content.mxf

nfs://projectserver/media/content.mxf
```

8.4 Capabilities Document

The capabilities document (section 5.11) allows implementations to find out which mappings, profiles, and protocols are supported before a transaction is initiated. Although most of the properties of a `capabilities` element are optional, it is recommended that implementations include as many as possible. It is also recommended that implementations support receiving a `qry_requestcapabilities` and sending an `rpl_capabilities` using MDP/XML/HTTP mapping, even if they will usually be using another mapping such as MDP/SOAP/HTTP.

9 Can MDP Carry Metadata?

The `file` element contains very little metadata about the file, just its name, size and MIME type. This was a deliberate design decision: only information that is relevant *to the transfer* of the file should be included. Furthermore, the standard prohibits implementations from adding their own metadata into the manifest document (this is sometimes called "dark metadata"). This means that for instance two organizations could not add a "contact email" property directly to the manifest (or if they do then it is no longer compliant with the standard). Implementations are also prohibited from adding new parameters to commands or queries.

However, there are scenarios when it would be useful to send metadata in advance of the main content of the delivery. For example by sending cataloguing keyword information about a set of video clips first, a user at the receiving end can start searching for content before it has all arrived. MDP supports two approaches to this:

The metadata is sent as a file (or files) within the transaction, scheduled or prioritized to be sent before the main content (see section 11).

The metadata is made available via a URL which is specified using a `details` property of a `file` element. This method has the potential advantage that this information can be used by the other agent to decide whether to accept or reject the file.

10 What Must Be Implemented and What is Optional?

Agents must implement at least one mapping and at least one profile. At the time of writing this means they must implement the MDP/XML/HTTP mapping and Basic Target Pull profile.

The MDP/XML/HTTP mapping requires support for both unencrypted HTTP/1.1 messages and HTTP/1.1 messages encrypted with TLS. (Most modern HTTP servers provide this as standard.)

The Basic Target Pull profile requires that agents support pull-mode transfer with both HTTP/1.1 and HTTP/1.1 encrypted with TLS.

Support for pause and resume functionality is optional with the Basic Target Pull profile, and agents are not required to signal when transfers have stalled, though it is recommended that they do so.

The protocol also includes reserved words to support new features, such as asynchronous notification of events, and use of encrypted XML, however, these cannot currently be used.

11 How Can Transfers Be Scheduled and Prioritized?

MDP provides support for delivery scenarios where time is critical, or where network bandwidth has to be reserved in advance. This includes the `startafter`, `finishbefore` and `priority` properties of `file` elements and the `availablefrom` and `availableto` properties of `transferoption` elements.

The simple negotiation procedure of the Basic Target Pull profile means that only the initiator agent can set these properties, and the following is recommended:

The initiator agent specifies the time window during which it wishes delivery to occur using `startafter` and `finishbefore`.

The initiator agent sets the `priority` properties to indicate any desired order of transfer (lowest number first). Files without `priority` properties will be transferred last.

`availablefrom` and `availableto` are only used if multiple `transferoptions` are present in the manifest document, but one or more of these can only be used during specified periods.

If it is necessary to reprioritize transfers after negotiation is completed, agents can pause and resume individual transfers as described in section 6.5.2.

12 How Can Agents Discover Each Other?

The standard assumes that at least one of the agents knows the other's message endpoint before the transaction (and before any exchange of capabilities documents). Any mechanism for finding these out automatically is outside MDP's scope.

In practice, one or more of the following methods might be used:

Agent information is exchanged manually by representatives of the organizations, possibly along with security information such as TLS certificates (see section 14).

A group of organizations who need to regularly exchange content sets an ad-hoc URL naming convention for message endpoints for use within that group, for example:

```
https://tvgroup.<organization_domain_name>/mdp/agent
```

A group of organizations who need to regularly exchange content provides a registry of message endpoints. This might be maintained by a neutral body or service provider.

The endpoint information is discovered automatically as part of another web service between the organizations, for example a business-to-business (B2B) web service that deals with the contractual side of a delivery. This might use a standard discovery web service such as UDDI (Universal Description, Discovery and Integration).

13 How Can Good Transfer Performance Be Obtained?

This question is out of scope of the MDP standard, which makes no performance requirements on implementations, and does not specify how any particular transfer protocol should be used (for instance it makes no requirements on how an HTTP server should be set up).

To achieve good performance, implementations typically might have to address some or all of the following factors:

The raw bandwidth of the link (including any switches, routers, etc.).

The network performance of the servers.

The amount and shape of traffic on the link.

The latency in the link. The throughput of TCP (and hence FTP and HTTP) decreases significantly once round-trip times become too high, an important factor on long-distance links. TCP tuning techniques can help considerably.

The read and write speeds of the media storage.

The choice of protocol, especially what type of encryption is used. A processor-intensive cipher may mean throughput is limited by available processing power. UDP-based transfer protocols can also be useful in some scenarios.

14 How Can Deliveries Be Secured?

14.1 General

MDP provide support for deliveries that are performed over insecure networks. The MDP/XML/HTTP mapping allows the HTTP messages to sent using Transport Layer Security (TLS – see RFC 4217). All compliant agents must implement this feature, and it is recommended that it be used for all transactions where security is a concern. When using HTTP/TLS, the message endpoints will have the `https:` scheme.

In addition, MDP allows the files to be transferred using secure protocols. The Basic Target Pull profile requires implementations to support HTTP/TLS, and the list in section 8.3 includes other secure protocols.

14.2 Authentication

It is recommended that when HTTP/TLS and other secure protocols are used, each agent provides authentication information. MDP does not define, or require, any particular way of doing this, and different methods may be better suited to different scenarios. In one approach, an agent's HTTPS server is configured to present its own server certificate (as is normal practice for secure web sites), but in addition to require that incoming HTTPS requests (i.e. MDP commands and queries) present an appropriate client certificate. Each agent can then authenticate the other's certificate. For best security it is recommended that any certificates are signed by a trusted certificate authority, or if this is not possible and self-signed certificates have to be used, that these have been previously exchanged in person by known representatives of each organization.

14.3 Integrity Check

It is also recommended that, when delivering files over insecure networks, the integrity of all files received be checked. Some transfer protocols support this directly, and MDP allows implementations to use this built-in functionality. However, HTTP/TLS does not, so MDP provides its own mechanism, in which the manifest document includes hash elements which provide a hash or digest value for each file. On receiving a file, an agent calculates its hash value and compares it with that in the manifest document. The Basic Target Pull profile requires that all implementations support the widely-implemented MD5 digest algorithm, but if a higher level of security is required for the integrity check, MDP also supports algorithms such as SHA-256.

MDP's built in integrity check can also be useful for non-secure deliveries, to provide a confirmation that the file has not been corrupted en route. An agent can also make use the hash value in the manifest document during negotiation: by comparing these with the hash value of files that the organization already possesses, unnecessary transfers could be avoided.

14.4 Manifest Document Properties

The manifest document includes a number of Boolean `file` properties to indicate:

whether a file must be sent using a secure transfer protocol (`mustencrypt`)

whether both ends must be authenticated for the transfer (`mustauthenticate`)

whether file integrity must be checked (`mustintegritycheck`)

It also includes `transferoption` properties declaring what actually will happen for a particular transfer option (`encrypt`, `authenticate`, `integritycheck`).

For the Basic Target Pull profile, all these properties are set by the initiator agent and so must be self-consistent.

15 Can Parts of Files Be Transferred?

MDP does not currently support partial file transfers; it is assumed that if there is a requirement to deliver a part of a file (for example between two timecodes), then some other process or system will already have created a new file with the required content.

16 How Can MDP Work through a Firewall or NAT?

The Basic Target Pull profile is intended for use where there is a direct network connection between the two agents. This is a reasonable assumption for many scenarios, where an external “dispatch area” might be separated from the internal operations of the organization (Figure 13). In such cases, files might be held in a separate dispatch storage store (and incoming files can be checked for viruses, etc., before being allowed into the working area).

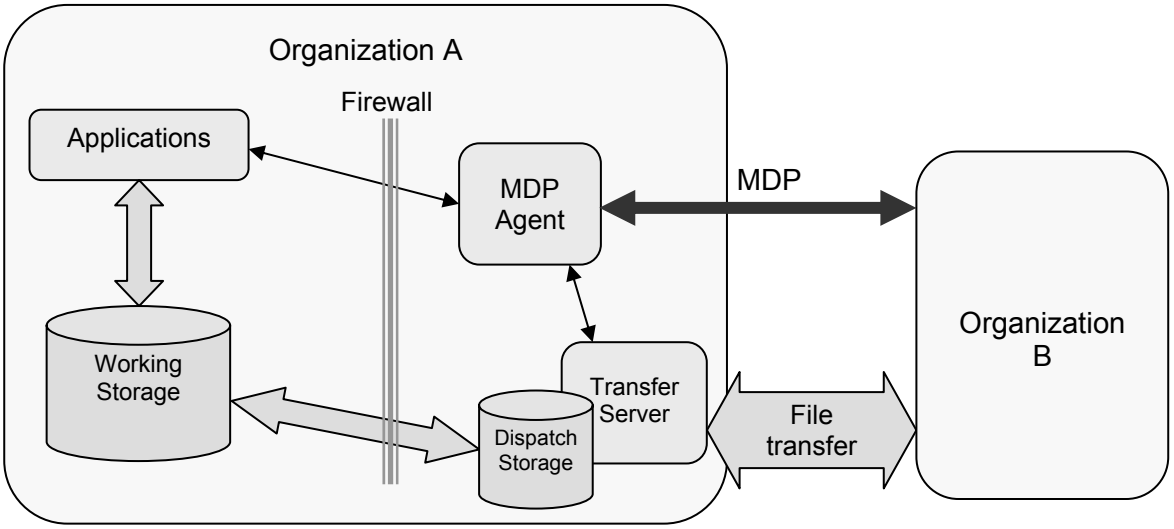

Figure 13 – Dispatch area outside firewall

However, other uses might require the MDP agent to be inside the firewall, or to be separated from the external network by a gateway that employs Network Address Translation (NAT). (Figure 14)

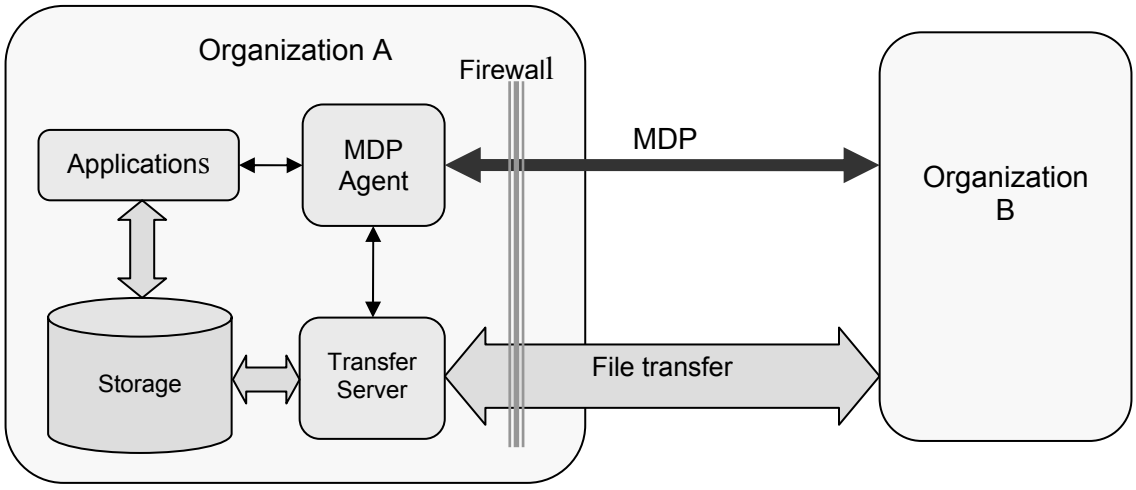

Figure 14 – MDP agent inside firewall

In the Basic Target Pull profile, both agents are required to send commands. With the MDP/XML/HTTP mapping this means that both agents are required to receive HTTP requests sent to their message endpoint. For this to work where NAT is being used, the NAT gateway will need to configured to forward these requests to the MDP agent. Where there is a firewall, this will also need to configured to allow incoming requests, which may otherwise be prohibited by the organization's security policy.

Another alternative is to set up a Virtual Private Network (VPN) connection between the organizations. This typically will also provide a secure connection, so a secure protocol may not be required.

Future MDP profiles may allow MDP to be used directly in such architectures, for example by making use of proxy servers.

17 How Can MDP Be Used within a Department or Facility?

MDP was developed for delivery between organizations and departments, typically over metropolitan- and wide-area networks, including the Internet. Of course, it could also be used for local-area transfers, including transfers within a department. In such cases, an organization identifier might be used to identify an editing system or a playout server. However, within a facility, transfer might occur by copying files over a shared file system rather than with HTTP, FTP, etc. MDP supports the use such cases by including the values `NFS` and `CIFS` in the protocol adminsitrative register Network File System, and the Common Internet File System.

Note: NFS is is a standardized protocol, but CIFS is not.

With shared file systems, there is another possibility: that no files are transferred, rather one system informs another that a file is available to be accessed directly. This is currently considered out of MDP's scope.

18 Can MDP Be Used for Streamed or Live Content?

MDP is currently intended for non-real-time file transfer. It is possible that the protocol might be used to initiate a streamed session; however, this has not been investigated.

19 Can MDP Be Used for Delivery to Multiple Recipients?

Although MDP can be used with multicast-capable transfer protocols such as FLUTE, the specification currently assumes only two agents are involved in a transaction, so multiple transactions would be required.

Future extensions to the protocol might, for example, allow a transaction in which the initial manifest document is sent to multiple target agents, each of which returns a manifest with its requirements, and then the initiator agent chooses a transfer option that is suitable for all; this might make use of multicast transfer. However, this is beyond the scope of this document.

Annex A
Requirements for Post-Production File Transfer

These requirements were drawn up by the Pro-MPEG Forum.

A.1 General

There is a general requirement to develop the following:

A common model for the secure, efficient and automatable exchange of media files between organizations over all types of IP network, in non-real time.

A set of recommended practices at each layer of the model using standard protocols and commodity technologies wherever possible.

A syntax and semantics to support the requirements for simple media file operations in professional audio/video applications with particular emphasis on the MXF (and AAF) file formats

A.2 Target Use Cases

The initial Pro-MPEG requirements used distribution of commercials for broadcast as its principal use case. The characteristics of this are as follows:

Many files involved, generally reasonably small.

Often several recipients, not generally more than 10.

As well as media files, approvals documents, digital signatures, and other documents specific to individual broadcasters requirements are also sent.

Within European countries, the number of sites is small, bandwidth is typically high (at least 10Mbit/s) and network distances are small (round trip time of less than 20ms). With the US the number of sites is large (1000s) and the bandwidth is variable (0.5 –10 Mbit/s) and network distances can be large (100 – 200ms).

Content is exchanged between sites within a large organization and between potentially competing organizations.

Transfers may be initiated manually, or may be triggered by an automated system.

Other use cases identified by Pro-MPEG included:

Delivery of program rushes (digital dailies) for editing

Delivery of program contributions

A.3 Simplicity

Any specifications should be easy to understand.

Implementations should be straightforward to install and manage.

It should be possible to start transferring with a new organization with minimal effort.

For manually-operated interfaces, media exchange should be as easily useable as email or "drag and drop".

A.4 Packaging

Users should be able to deal with the delivery of packages of files rather than having to worry about all the individual file transfers. It should be possible to readily determine whether a particular package has:

been agreed for transfer (or refused)

started transferring

completed successfully (or failed)

It should also be possible to "drill down" into the package to see the state of each individual transfer.

A.5 Content

A wide range of numbers of files and file sizes might be sent in a delivery. At one extreme, a two-hour film at 4K resolution might be sent uncompressed in a single file, while at the other extreme, several files might be sent per frame.

Material, metadata and other files, such as contract and rights information and dope sheets, will be sent.

Content only has to be uniquely identified between organizations rather than using a globally unique identifier (UMID, etc).

It should be possible for an organization to decide whether it wishes to receive a particular piece of content before it is transferred.

It should be possible for an organization to decide which version(s) of an item should be sent. This could include different versions with different editorial content or different formats.

There may be cases where the content to be transferred is created on demand following a request from the recipient.

A.6 Transfer

No single transfer protocol should be mandated.

Standards-based transfer protocols are preferred, but it is recognized that users will continue to want to use proprietary delivery solutions in particular cases, so implementations should be capable of initiating transfers using a range of protocols.

Some types of partial transfer may be required, including byte-ranges, time-ranges, individual tracks (typically from an MXF files), and transfer of header metadata.

Both push-model and pull-model transfer are identified as being required. Pull-model transfer is seen as being of particular importance in use cases where the recipient has tight timescales as it may allow better control over what it receives and when it receives it.

The initial user requirement is for point-to-point transfer, but point-to-multipoint transfer may also be required, in which the same transfer/security/reliability requirements apply to each recipient. No part of the system should unnecessarily preclude supporting point-to-multipoint applications.

A.7 Security

It is assumed that business arrangements already exist to permit a transfer. These are outside the scope of the group's work.

It should be possible to authenticate endpoints, using standard mechanisms.

It should be possible to authorize, encrypt and authenticate file transfers, using standard mechanisms.

It should be possible to check the integrity of data that has been transferred.

Both unauthorized reading and unauthorized writing should be considered.

It should be possible to use these mechanisms in an automated way (e.g. not requiring user password entry), and to interface with systems that manage security policies.

These security mechanisms are independent of and must not interfere with any other security mechanism such as file-content encryption, watermarking etc. or network security mechanisms.

Only the security of the exchange is considered; no assumptions or requirements are made regarding how organizations secure content within their own infrastructure.

It is assumed that organizations will adopt best practice for network security, e.g. applying patches, taking action against denial of service attacks, etc.

Solutions should be capable of working in the context of an organization's security infrastructure, e.g. where NAT, firewalls, content filtering and/or virus filtering are used.

A.8 Performance and Reliability

It should be possible to prioritize the utilization of the available bandwidth.

It should be possible to make efficient use of the end-point resources and link capacity available.

It must be possible to achieve and confirm bit-perfect copy of the transferred data.

In the case of failed transfers, implementations should attempt to recover and, when not possible, report the failure and its details to the end user or application.

A.9 Control and Monitoring

It should be possible to monitor use of network and end-point resources and transfer performance.

Interfaces with standards-based facility and network management systems for control and monitoring should be available (e.g. using SNMP MIB).

A.10 Standards

Widely used standards defined by accredited standards organizations (IETF, ISO, SMPTE, W3C, etc.) should be used wherever possible.

Annex B
Glossary

The following words and phrases are defined normatively in SMPTE 2032-1 and are repeated here for information.

Agent: A software entity that acts on behalf of an organization to implement the Media Dispatch Protocol.

Command: A message that typically causes change within a transaction or sends information.

Confirmation: A message that provides an immediate positive or negative acknowledgement of a command.

Data structure: A defined piece of information that may be carried within a message. May be an element, property or list.

Dictionary: Allowed values of properties.

Element: A data structure containing other elements, lists or properties.

File: A unit of content, typically a stored media file, though files may be dynamically generated.

Initiator agent: The agent that sends a message to initiate a transaction.

List: A data structure containing a single unordered set of properties or elements of the same type.

Manifest document: A representation of the details and state of a transaction. Takes the form of an element.

Mapping: A specification for how MDP messages and data structures shall be represented on the network.

Media Dispatch Protocol: A standard mechanism for two organizations to initiate a delivery, agree on the details of how it should happen, and manage the delivery.

Message: A message passed between agents. May be a command, confirmation, query or reply.

Message endpoint: A URL at which an agent receives a message.

Organization: A company, corporation, department, etc. that requires delivery of files to or from another organization.

Profile: A specification of a subset of MDP intended to aid interoperability. Defines which messages and data may and may not be used, and defines allowed sequences of messages and actions.

Property: A data structure containing a single simple value.

Pull-mode transfer: A transfer initiated from the receiving end.

Push-mode transfer: A transfer initiated from the sending end.

Query: A message that allows an agent to request information from another agent.

Reply: A message in immediate response to a query.

Target agent: The agent that receives a message to initiate a transaction.

Transaction: The sequence of steps undertaken by two agents while negotiating and performing the transfer of a set of files.

Transfer endpoint: A sending or receiving URL used for a file transfer.