216217 P2P-Next
D5.2.1d
Content packaging, ingestion, and adaptation

Contractual Date of Delivery to the CEC:	31st December 2011
Actual Date of Delivery to the CEC:	31st December 2011
Author(s):	Michael Eberhard (UNIKLU)
Participant(s):	Chris Needham (BBC), Dusan Gabrijelčič (JSI)
Workpackage:	5
Est. person months:	
Security:	PU
Nature:	P
Version:	1.9
Total number of pages:	74

Abstract:

This deliverable contains descriptions of the solutions for content packaging, content ingestion, and content adaptation utilized within the NextShare system.

Keyword list: Content packaging, content ingestion, content adaptation

Version History

Version	Author	Description	Date
0.1	Michael Eberhard	Initial Draft	09-01-2009
1.0	Michael Eberhard	Added BBC's content ingestion approach, an alternative for content packaging and more details to content adaptation	27-02-2009
1.1	Dušan Gabrijelčič	Added JSI content ingestion approach and more details on its implementation	28-02-2009
1.2	Michael Eberhard	Updated content packaging part according to the architecture for M24	30-10-2009
1.3	Michael Eberhard	Updated content adaptation part, various minor updates	07-12-2009
1.4	Michael Eberhard	Updated content ingestion part, minor updates according to feedback from internal review	21-12-2009
1.5	Michael Eberhard	Updated content adaptation	09-11-2010
1.6	Michael Eberhard	Update of all sections for M36	07-12-2010
1.7	Michael Eberhard	Updates according to feedback from internal review	16-12-2010
1.8	Michael Eberhard	Initial version for M48	07-11-2011
1.9	Michael Eberhard	Final version	19-12-2011

Executive Summary

This document describes the content packaging, content ingestion, and content adaptation solutions utilized within *NextShare*. A more detailed description of each of the three parts is provided in the following.

The content packaging provides a solution to packetize the content, the metadata and possibly additional metadata for the content items. To ensure backwards compatibility to older Bittorrent clients, the top-level description of every content item is a torrent file. In addition to the usual torrent attributes, the file contains the core metadata as specified in [1] and possibly references to optional metadata or other data related to the content item. The additional data referenced can be provided by either the peer-to-peer system or alternatively by servers. The main reasons for selecting this packaging solution were to keep the top-level torrent file rather small by only storing the essential data within the torrent file and to still provide all the necessary metadata attributes needed for search (which are provided by the core metadata).

The content ingestion part describes tools needed to ingest professional content as well as user-generated content into the *NextShare* system. As the first trials of the *NextShare* system will only use professional content, this version of this deliverable only provides two ingest mechanisms for professional content providers, i.e., the content ingestion approach utilized by BBC and JSI.

The content adaptation solution provides a codec-agnostic adaptation framework for scalable video content. This adaptation solution is developed as the joint work of WP4, WP5, and WP6. In this deliverable, the piece-picking algorithm, which provides a mechanism to download the pieces of the highest possible quality while still ensuring that the pieces are downloaded in time for playback, is described in detail. Additionally, the context-related metadata, which reflect the capabilities of the user terminals as well as the user preferences, are described.

The main changes to the previous version of this deliverable include:

- Section 2 was not updated, as the content packaging solution has not been modified. Only Section 2.2 has been updated to point out that the software package is not provided with this deliverable anymore, but has been integrated into *NextShare* and is part of the NextShare software package [7].
- Section 3.1 was not updated, as BBC's content ingest solution has not been modified.
- Section 3.2 was updated to describe the updates of JSI's content ingest solution (in particular, Sections 3.2.5 and 3.2.6 have been updated). Again, the software package is not provided with this deliverable anymore, but has been integrated into *NextShare* and is part of the NextShare software package [7].
- Section 4.3.2 has been updated to document the new work on the piece-picking algorithm.

Table of Contents

Index of Figures

Index of Tables

1 Introduction

The present document describes the content packaging, content ingestion and content adaptation mechanisms applied within the *NextShare* system.

The content packaging part provides a framework for packetizing of all media and metadata files. Additionally, backwards compatibility to other *Bittorrent* clients should be assured by still providing a torrent file compatible with the *Bittorrent* protocol. Thus, the main file of a content package is the torrent file, which contains the core metadata, that are required for searching. All other metadata are referenced from this top-level torrent file. The main reason for utilizing this mechanism is that the torrent file should be as small as possible, as it needs to be distributed to all peers, but should still contain sufficient data to be able to discover the desired content based on the information within the torrent file.

The content ingestion part describes tools needed to ingest professional content as well as user-generated content into the *NextShare* system. As the first trials of the *NextShare* system will only use professional content, this version of this deliverable only provides two ingest mechanisms for professional content providers, i.e., the content ingestion approach utilized by BBC and JSI. Based on these two content ingestion mechanisms, a prosumer tool will be developed that allows third party content providers to join the *NextShare* trials. The content ingestion approaches are described in Section 3.

The content adaptation part describes the integration of scalable content into the *NextShare* system. As the users of the *NextShare* system use heterogeneous terminals and are connected to networks with different bandwidth capabilities, it is desirable to provide these users with multimedia streams in different qualities. Additionally, the network conditions in our system are not always stable and due to changes in the network connections a change in quality of content might be desirable. However, if one multimedia stream would be distributed in different encodings, it is highly unlikely that all versions would be equally distributed among the users of the *NextShare* system and a switching between the different versions would be rather difficult. A feasible alternative is providing a scalable bitstream, which provides the bitstream once in best quality but can be easily adapted to a lower quality. The integration of such a scalable codec into the *NextShare* system is described in Section 4.

2 Content Packaging

The task 5.2.1, content packaging, provides a framework for packetizing the metadata and media resources and to reference them from the top-level torrent file to enable easy content ingestion. A main requirement for this task is that we want to be backwards compatible with other *Bittorrent* clients. Thus, a torrent file compatible with the *Bittorrent* protocol needs to be provided as top-level description. However, the torrent files contain only a small part of the metadata needed to represent rich media content. Therefore, a packaging solution for the metadata as defined in [1], the media resources and possibly additional data like LIMO data [2] needs to be provided.

Based on these considerations, a packaging solution utilizing MPEG-21 Digital Item Declaration (DID) [3] is proposed. An overview of this packaging solution is illustrated below.

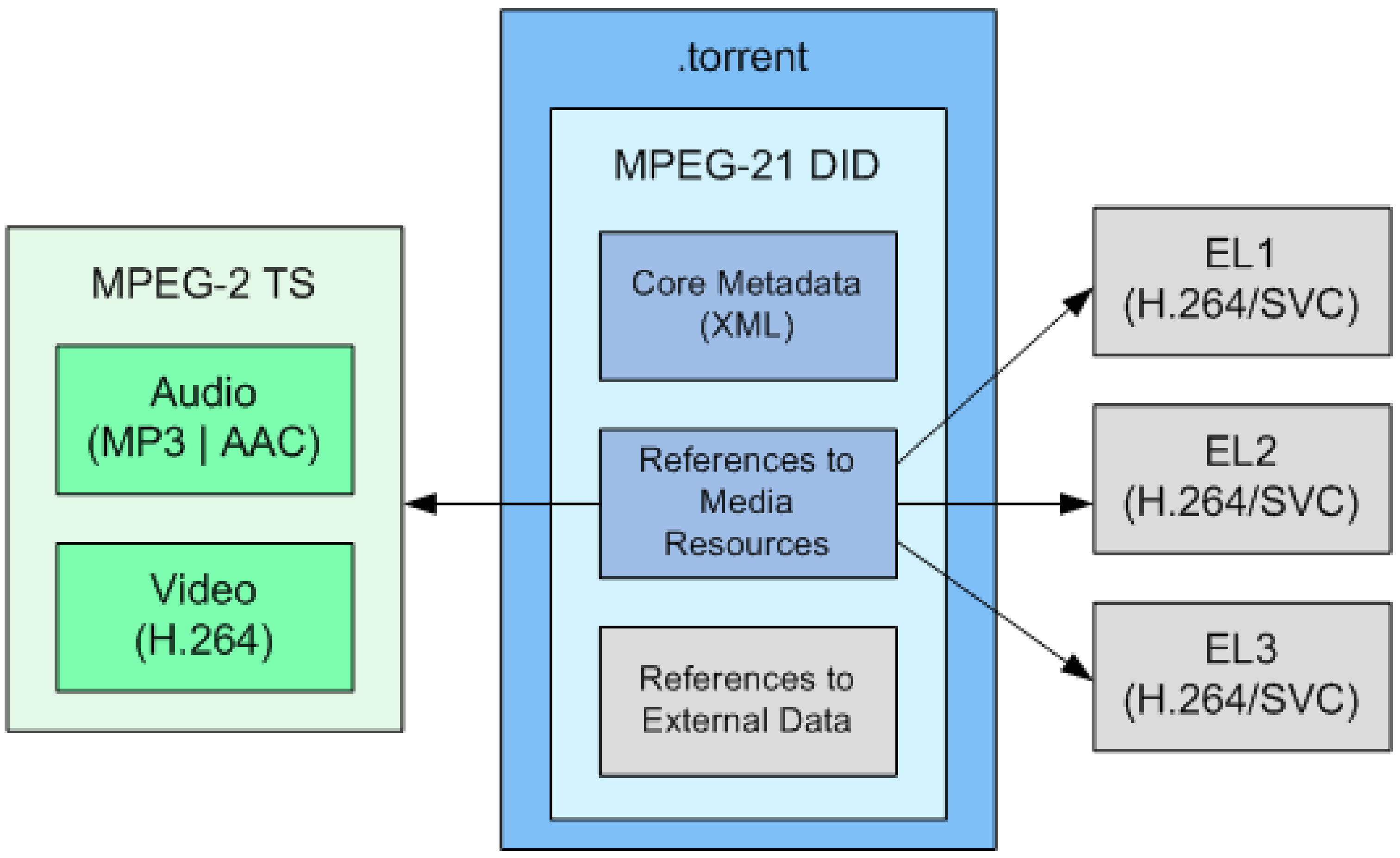


Figure 1: Packaging Solution Overview

The top-level torrent file contains the references and hash values for the media resources, i.e., for the MPEG-2 Transport Stream containing the H.264 video and the audio content, and possibly for the Scalable Video Coding (SVC) enhancement layers [4], if the video content is scalable. Additionally, the torrent file contains the top levels of the MPEG-21 DID description, which might be encoded in binary XML (or just provided as plain XML). The DID included in the torrent file contains the core metadata from the P2P-Next Rich Metadata specification and references to other metadata and resources. The main reason for storing only the core metadata directly in the torrent file is that we want to keep the size of the torrent file as small as possible (by referencing the other data) and still provide sufficient data to base the search on the content of the torrent file. A more detailed description of the structure of the DID is provided in the next section.

2.1 Usage of MPEG-21 Digital Item Declaration

This section describes the packaging of an P2P-Next Item utilizing an MPEG-21 DID. The P2P-Next Item contains all media resources, metadata, and possible additional data related to a single digital object. The P2P-Next Item is described by an DID which structures content and metadata. The DID contains relevant metadata and provides references to media resources and distributed metadata. The core metadata from the P2P-Next Rich Metadata (RM) specification are directly included in the top levels of the DID. All optional RM such as payment, advertisement, and scalability is referenced via XML Inclusions (Xinclude), as these metadata are only required for specific services and might be provided, e.g., on a secure server for payment. The data for LIMO is also referenced through XInclude. The LIMO data is based on HTML 5 and can contain HTML or Javascript code, style sheets or media resources. As it is difficult to include media resources into XML files (base64 encoding is not a feasible solution for large media resource files) the LIMO content is packetized into the MPEG-21 file format [5], which contains another DID that references all the LIMO content within the MPEG-21 file. In this way, the part of the DID that is stored directly in the torrent file is kept as small as possible while retaining a complete DID structure and conveying the RM core metadata inside the torrent file for increased search performance. Thus, there are two (or more) DIDL documents.

The main document conveys the overall structure of the P2P-Next Item and core metadata. It is stored inside the torrent file. The additional document is stored in the XML-Box of an MPEG-21 file and can contain the additional RM as well as data for LIMO. The torrent file references that MPEG-21 file. The main document references parts of the additional document as described above. However, the conceptual DID model for the P2P-Next Item should be seen as one entity, only its physical representation is split into two (or more) documents. Please note that the usage of an MPEG-21 file is not mandatory. The second DID document does not necessarily need to be stored within an MPEG-21 file. If it is more advantageous to distribute the LIMO content and the additional metadata in separate files, the DID just needs to reference these files and the can be provided separately through the *NextShare* system or on servers. The MPEG-21 file just provides one possibility to store all the additional data together, if such a packaging mechanism is desired.

Figure 2 outlines the DID for a P2P-Next Item. The building blocks of a DID are shown on the left side. On the right side of the figure, shapes with a dotted outline indicate data that is not directly contained within the main DIDL document in the torrent file, but is rather being referenced for the concern of decreased size of the torrent file.

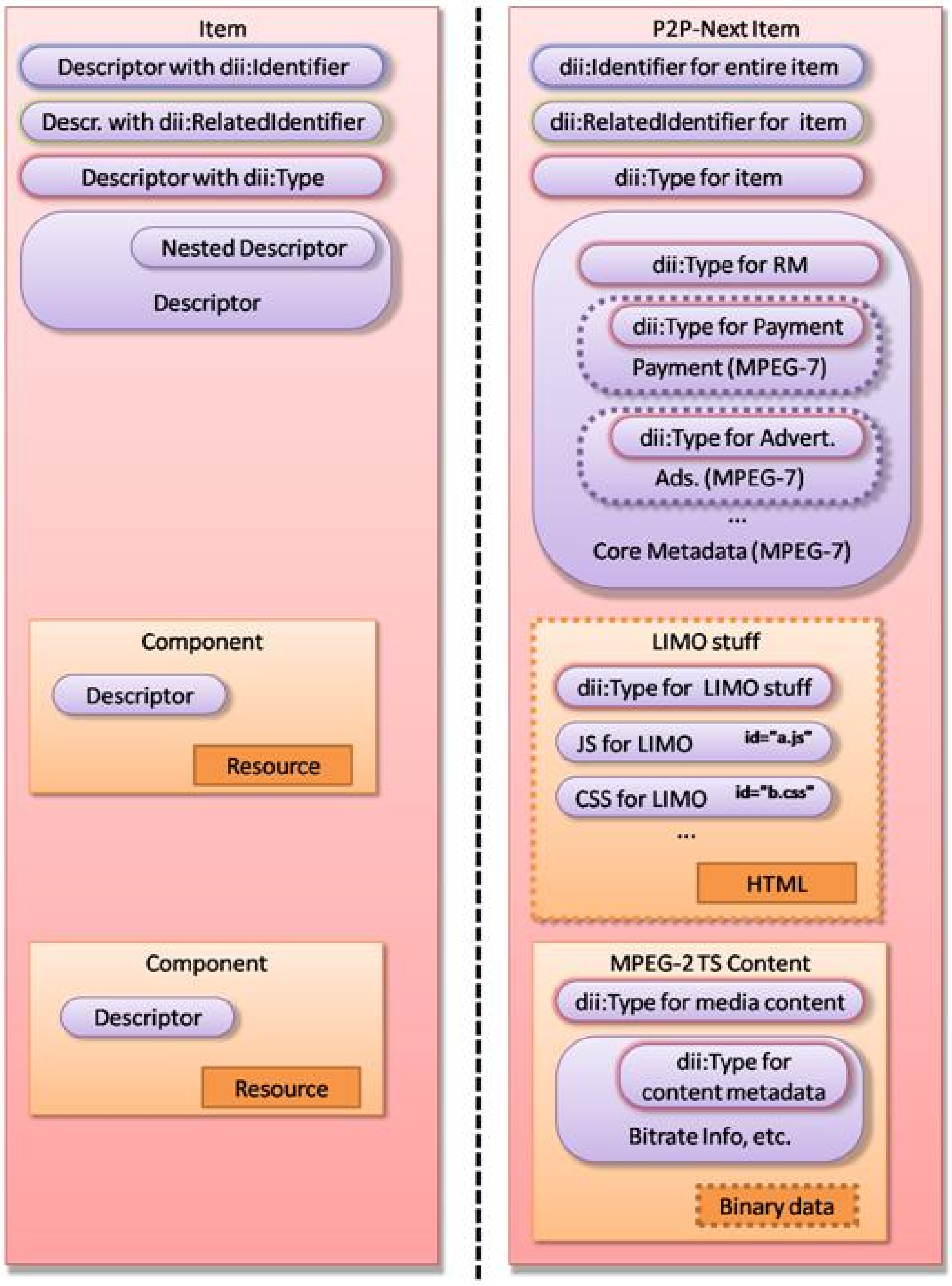


Figure 2: MPEG-21 DID Example

The P2P-Next Item is represented by an Item. The dii:Identifier is an MPEG-21 Digital Item Identifier (DII) for the entire item, e.g., the Uniform Resource Name (URN) "urn:p2p-next:item:bbc-bbcone-b00n9p5x". It is enclosed by a Statement inside a Descriptor. Note that Statements are not shown in this figure for the sake of simplicity. The next Descriptor contains a Statement with a dii:RelatedIdentifier which allows identifying the underlying work described by a Digital Item (DI). In this case, the RelatedIdentifier defines an "isAbstractionOf" relation to the underlying media content, e.g., identified by "urn:bbc-bbcone-b00n9p5x". This underlying media content is independent of the P2P-Next system. The dii:Type identifies the structure of this DID. It is set statically to the URN "urn:p2p-next:type:item:2009" and, thus, determining the position within the DID and which building blocks are allowed. The structure of this DID is defined as part of this deliverable.

The RM is represented following the Rich Metadata specification [1]. The Descriptor for the RM is structured as follows. The core metadata is contained within a Statement which is typically the last element in this Descriptor. Some nested Descriptors precede that Statement. The first one contains the dii:Type for the core metadata. This Type is set statically to "urn:p2p-next:type:rm:core:2009" and identifies the structure of the RM core metadata within a P2P-Next Item as defined by [1].

All additional RM is contained in further nested Descriptors which are referenced via XInclude. For each Descriptor, an xi:include element points to Descriptor in the additional DIDL document. Each of these Descriptor has a dii:Type to identify its structure and consequently its purpose. The Types are listed in Table 1. Each Descriptor contains a Statement which conveys the RM representation of the appropriate RM part. More details on the URNs utilized within *NextShare* are provided in Annex A.

Additional Metadata	**URN for dii:Type**
Payment	urn:p2p-next:type:rm:payment:2009
Advertisement	urn:p2p-next:type:rm:advertisement:2009
Scalability	urn:p2p-next:type:rm:scalability:2009

Table 1: URN for dii:Type

There are two Components in the DID for the P2P-Next Item. The first Component contains all data for LIMO. It is stored in the additional DIDL document inside the separate MPEG-21 file. It is included into the main document by means of an xi:include element. The dii:Type in its first Descriptor is set statically to the URN "urn:p2p-next:type:limo:2009". Further Descriptors contain resources required by the actual LIMO resource (i.e., the HTML content). These resources may be JavaScript files, Cascading Style Sheets (CSS) as well as JPEG or PNG images. Each Descriptor contains an id attribute, uniquely identifying that resource within the P2P-Next Item. It is proposed to use the original file name of a resource in all lower-case characters for the id value if applicable. The resources with text content are contained directly in a CDATA section of the Resource. On the other hand, resources with binary content, such as JPEG images, are bundled in the MPEG-21 file. The Resource of the Component contains the HTML page representing the LIMO content. As the required resources are not in actual files but rather in one MPEG-21 wrapper file, all references to the original files have to be replaced in the HTML document by the corresponding ids. For example, the reference to the file "script.js" would be replaced by "#script.js", a reference to the Descriptor with the id "script.js". These replacements could be accomplished by means of an XSL Transformation (XSLT). The HTML document for LIMO is contained in the CDATA section of the Resource.

The second Component in this Item represents the actual media content. The media content shall typically be packed into an MPEG-2 TS. The Component contains a Descriptor with a dii:Type which is set to "urn:p2p-next:type:content:2009". Furthermore, a second Descriptor may be present, conveying technical metadata about the TS (such as bitrate, size, etc.). The actual binary data of the TS is referenced through a Resource within the Component.

An example of how such a DID can be composed is illustrated below.

```
<didl:DIDL xmlns:didl="urn:mpeg:mpeg21:2002:02-DIDL-NS"
xmlns:dii="urn:mpeg:mpeg21:2002:01-DII-NS" xmlns:xi="http://www.w3.org/2001/XInclude"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:mpeg:mpeg21:2002:02-DIDL-NS didl.xsd
urn:mpeg:mpeg21:2002:01-DII-NS dii.xsd">
     <didl:Item>
          <didl:Descriptor>
```

```
                <didl:Statement mimeType="text/xml">
                     <dii:Identifier>
                          urn:p2p-next:item:bbc-bbcone-b00n9p5x
                     </dii:Identifier>
                </didl:Statement>
           </didl:Descriptor>
           <didl:Descriptor>
                <didl:Statement mimeType="text/xml">
                     <dii:RelatedIdentifier relationshipType=
                     "urn:mpeg:mpeg21:2002:01-RDD-NS:IsAbstractionOf">
                          urn:bbc-bbcone-b00n9p5x
                     </dii:RelatedIdentifier>
                </didl:Statement>
           </didl:Descriptor>
           <didl:Descriptor>
                <didl:Statement mimeType="text/xml">
                     <dii:Type>urn:p2p-next:type:item:2009</dii:Type>
                </didl:Statement>
           </didl:Descriptor>
           <didl:Descriptor>
                <!-- rich metadata -->
                <didl:Descriptor>
                     <didl:Statement mimeType="text/xml">
                          <dii:Type>urn:p2p-next:type:rm:core:2009</dii:Type>
                     </didl:Statement>
                </didl:Descriptor>
                <!-- references to additional RM -->
                <xi:include href="put_here_URI_of_m21_in_torrent"
                xpointer="rm.payment" />
                <xi:include href="put_here_URI_of_m21_in_torrent"
                xpointer="rm.advertisement" />
                <xi:include href="put_here_URI_of_m21_in_torrent"
                xpointer="rm.scalability" />
                <didl:Statement mimeType="text/xml">
                     <!-- P2P-Next RM -->
                </didl:Statement>
           </didl:Descriptor>
           <!-- References to LIMO -->
           <xi:include href="put_here_URI_of_m21_in_torrent" xpointer="limo" />
           <didl:Component>
                <didl:Descriptor>
                     <didl:Statement mimeType="text/xml">
                          <dii:Type>urn:p2p-next:type:content:2009</dii:Type>
                     </didl:Statement>
                </didl:Descriptor>
                <didl:Descriptor>
                     <didl:Descriptor>
                          <didl:Statement mimeType="text/xml">
                               <dii:Type>
                                    urn:p2p-next:type:content:metadata:2009
                               </dii:Type>
                          </didl:Statement>
                     </didl:Descriptor>
                </didl:Descriptor>
                <didl:Resource mimeType="video/mp2t"
                ref="put_here_URI_of_mp2ts_in_torrent"/>
           </didl:Component>
      </didl:Item>
</didl:DIDL>
```

The second example shows the additional DIDL document that is included within the MPEG-21 file for additional metadata and LIMO content. This DIDL document also contains Descriptors for the dii:RelatedIdentifier and dii:Type. The dii:RelatedIdentifier references the main DIDL document through a "IsPartOf". The dii:Type is set to "urn:p2p-next:type:item:additional:2009".

```
<didl:DIDL xmlns:didl="urn:mpeg:mpeg21:2002:02-DIDL-NS"
xmlns:dii="urn:mpeg:mpeg21:2002:01-DII-NS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:mpeg:mpeg21:2002:02-DIDL-NS didl.xsd
urn:mpeg:mpeg21:2002:01-DII-NS dii.xsd">
     <didl:Item>
          <didl:Descriptor>
               <didl:Statement mimeType="text/xml">
                    <dii:RelatedIdentifier relationshipType=
                    "urn:mpeg:mpeg21:2002:01-RDD-NS:IsPartOf">
                         urn:p2p-next:item:bbc-bbcone-b00n9p5x
                    </dii:RelatedIdentifier>
               </didl:Statement>
          </didl:Descriptor>
          <didl:Descriptor>
               <didl:Statement mimeType="text/xml">
                    <dii:Type>urn:p2p-next:type:item:additional:2009</dii:Type>
               </didl:Statement>
          </didl:Descriptor>
          <!-- Additional RM -->
          <didl:Descriptor id="rm.payment">
               <didl:Descriptor>
                    <didl:Statement mimeType="text/xml">
                         <dii:Type>urn:p2p-next:type:rm:payment:2009</dii:Type>
                    </didl:Statement>
               </didl:Descriptor>
               <didl:Statement mimeType="text/xml">
                    <!-- P2P-Next RM on payment -->
               </didl:Statement>
          </didl:Descriptor>
          <didl:Descriptor id="rm.advertisement">
               <didl:Descriptor>
                    <didl:Statement mimeType="text/xml">
                         <dii:Type>urn:p2p-next:type:rm:advertisement:2009</dii:Type>
                    </didl:Statement>
               </didl:Descriptor>
               <didl:Statement mimeType="text/xml">
                    <!-- P2P-Next RM on advertisement -->
               </didl:Statement>
          </didl:Descriptor>
          <didl:Descriptor id="rm.scalability">
               <didl:Descriptor>
                    <didl:Statement mimeType="text/xml">
                         <dii:Type>urn:p2p-next:type:rm:scalability:2009</dii:Type>
                    </didl:Statement>
               </didl:Descriptor>
               <didl:Statement mimeType="text/xml">
                    <!-- P2P-Next RM on scalability -->
               </didl:Statement>
          </didl:Descriptor>
          <didl:Component id="limo">
               <!-- LIMO content -->
               <didl:Descriptor>
                    <didl:Statement mimeType="text/xml">
                         <dii:Type>urn:p2p-next:type:limo:2009</dii:Type>
                    </didl:Statement>
               </didl:Descriptor>
               <didl:Descriptor id="jquery-1.3.2.min.js">
                    <didl:Component>
                         <didl:Resource mimeType="text/javascript" >
                              <![CDATA[
                                   javascript:...
                              ]]>
                         </didl:Resource>
                    </didl:Component>
               </didl:Descriptor>
               <didl:Descriptor id="jquery.jcarousel.css">
                    <didl:Component>
                         <didl:Resource mimeType="text/css">
                              <![CDATA[
                                   css:...
                              ]]>
                         </didl:Resource>
                    </didl:Component>
               </didl:Descriptor>
               <didl:Descriptor id="logo.jpg">
                    <didl:Component>
                         <didl:Resource mimeType="image/jpeg"
                         ref="urn:p2p-next:item:bbc-bbcone-b00n9p5x:additional:1" />
                    </didl:Component>
               </didl:Descriptor>
               <didl:Resource mimeType="text/html">
                    <![CDATA[
                         <html>...
                    ]]>
               </didl:Resource>
          </didl:Component>
     </didl:Item>
</didl:DIDL>
```

2.2 Packaging API

The Packaging API is used for two major purposes: Firstly, during the creation of the top-level torrent file, the API is utilized to create the MPEG-21 DID describing the content package. Secondly, during the distribution of the content in the *NextShare* system, the API is utilized to access the different parts of the DID such as the core metadata. The API is implemented in C++ using the CubeWerx BXML library [6] and provides Python bindings for usage with the content ingestion tools implemented in Python (see Section 3.2.3).

Tha Packaging API has been integrated into *NextShare* and thus no software package is provided with this deliverable. Instead, the Packaging API is part of the *NextShare* deliverable [7].

The Packaging API for *NextShare* provides two major interfaces. The *DIDCreator* allows to create MPEG-21 DIDs containing the core metadata, references to optional metadata and to media resources, while the *DIDParser* allows to extract the content from DIDs. The methods of both interfaces are described in detail below.

Method	Description
DIDCreator	
Void **addIdentifier**(char* id)	Adds an identifier with the specified ID to the DID document. The ID specified in this method is used to identify the item described by the DID in the *NextShare* system.
void **addDescriptorWithUrn**(char* urn)	Adds a descriptor with the specified URN as *dii:Type* to the DID document. This method is used to specify the type of content in the subsequent Statement (e.g., urn:p2p-next:type:rm:core:2009 for P2P-Next core RM).
void **addXInclude**(char* href, char* xpointer)	Adds an *xi:include* element to the DID document, which is used to reference external (meta-)data. This method can be used to add references to optional external metadata.
void **addStatement**(char* mimeType, char* text)	Adds a Statement with the specified content to the DID document. This metadata can be used to add the core RM content to the DID document.
void **openDIDL**()	Creates a new DIDL tag.
void **closeDIDL**()	Closes a DIDL tag.
void **openItem**()	Creates a new Item tag.
void **closeItem**()	Closes an Item tag.
void **openDescriptor**()	Creates a new Descriptor tag.
void **closeDescriptor**()	Closes a Descriptor tag.
DIDParser	
const CW_XML_NODE* **getContentByDiiType**(CW_XML_SCAN* xmlScan, const char* diiType)	Extracts the content from the DID document based on the specified dii:Type.
const CW_XML_NODE* **getContentByID**(CW_XML_SCAN* xmlScan, const char* id)	Extracts the content from the DID based on the Descriptor's ID. The Descriptor ID is only utilized for LIMO content.
char* **getXIncludeHref**(char* xpointer)	Extracts the href for an external resource based on the specified xpointer.

Table 2: DID API Description

3 Content Ingestion

The task 5.2.2, content ingestion, provides the tools needed to ingest content from professional content producers as well as user-generated content. As the first trials of the *NextShare* system will only contain content from professional content producers, the focus of this deliverable is on professional content ingestion. Thus, the content ingestion solution provided by the BBC is described in the next section. The content ingest solution provided by the JSI is described in the section 3.2. The solutions target similar system tasks but are complementary in their design and implementation. Finally, a prosumer tool that allows also third party content providers to join our *NextShare* trials should be developed.

The proposal outlined in sections 3.1.1 and 3.1.2 including related subsections was superseded by an architecture proposed and implemented by ULANC that transcodes video received from broadcast, combining this with metadata feeds from the BBC, and seeding this to servers hosted at ULANC. This is described fully in [8], section 2.2.5, Content Ingest. The original description is retained in this document for historical interest.

3.1 BBC content ingest and publishing solution

3.1.1 Aims and requirements

The aim of the BBC ingest and publishing system proposed here was to make live and on-demand content available via *NextShare*.

The system must be capable of large-scale ingest of content, in multiple formats from multiple sources, with minimal human interaction.

Metadata ingest and publishing must be coordinated with content ingest and publishing: this process is outlined in section 4 of [1].

Alternative solutions are described in section 3.1.3; the BBC's proposed architecture is described below.

3.1.1.1 BBC proposed architecture

The proposed architecture to publish content via *NextShare* and HTTP for the BBC is summarised in the following points. This proposal was superseded by an architecture operated by ULANC):

•Live and on-demand assets provided via P2P and HTTP.

•Automatic creation and HTTP publication of torrents.

•BBC-operated P2P supernode to seed BBC content for the duration of the official availability window (superseded by ULANC-operated supernode).

•Torrents to point to appropriate trackers (or DHT-like schemes) as advised by WP4.

•Comprehensive content discovery feeds, using an ATOM Discovery Feed Tree (as outlined in section 4 of [1]).

•LIMO applications available via both HTTP, and automatically packaged in the torrents they relate to (superseded by decision not to package LIMO content in the torrents. NextShare is most effective as distributing larger items of content and its effectiveness is reduced by adding large quantities of smaller files).

•Possible future transcode service to create alternative encodings of assets.

Figure 3 shows the high-level architecture of the BBC content ingest and publishing service.

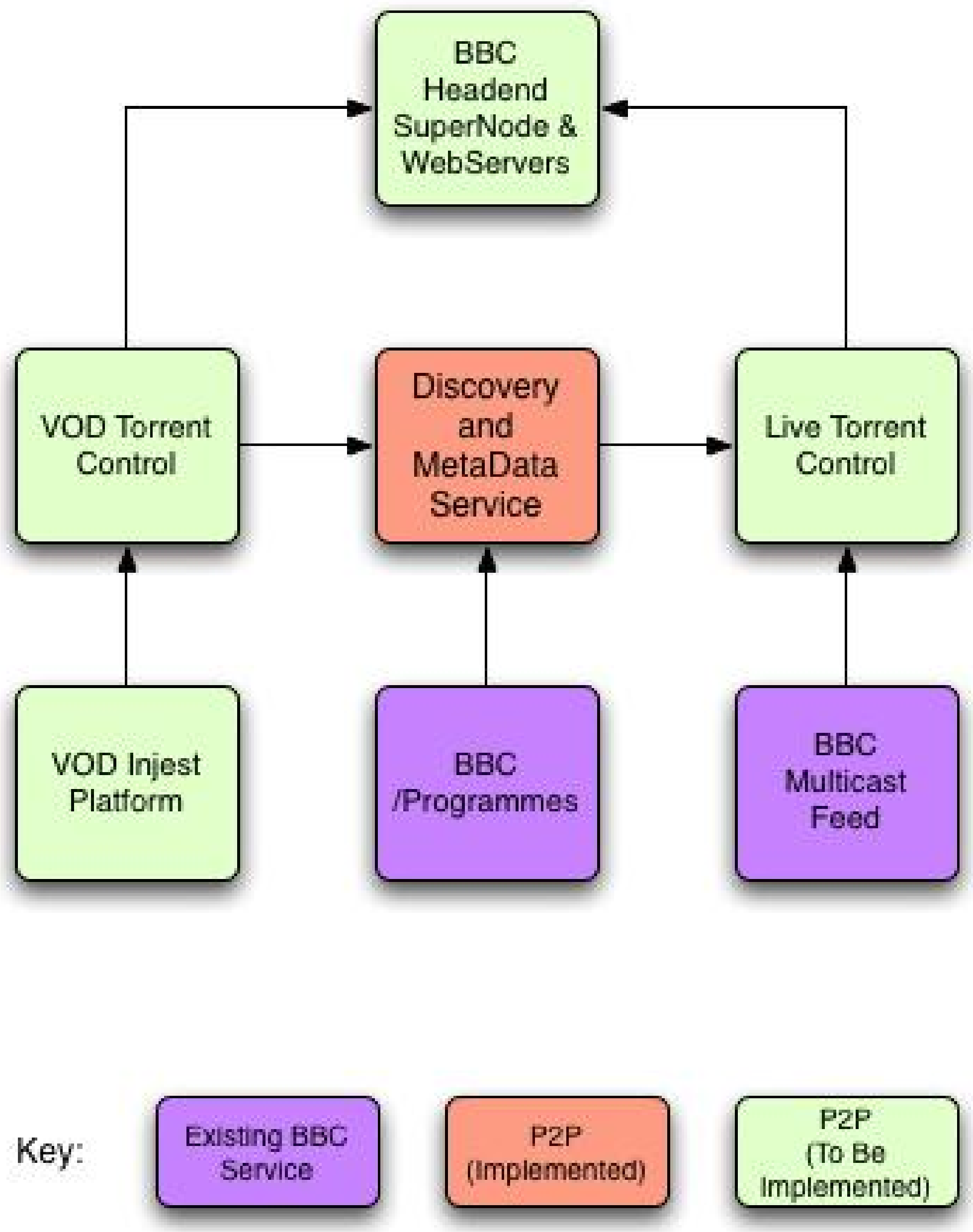


Figure 3: BBC content ingest and publishing service

3.1.1.1.1 Discovery and Metadata Service

The anticipated BBC solution for discovery and content metadata feeds is described in sections 4.1 and 4.2 of [1].

3.1.1.1.2 VOD Torrent Control

The VOD Torrent Control service was intended to act as a data store for VOD data, and control the availability windows for VOD data. This has been superseded by a solution implemented by ULANC.

3.1.1.1.3 VOD Ingest Platform

The VOD Ingest Platform was intended to be the interface used to upload VOD data bundles with associated metadata and set the availability windows. It was anticipated that VOD data bundles may include multiple assets including LIMO applications and that the VOD Ingest Platform may also perform transcoding of assets to appropriate formats as required.

3.1.1.1.4 Live Torrent Control

The Live Torrent Control service was to generate the Live Torrent Information for BBC services using the data generated from the discovery service to build appropriate torrents, i.e. including associated LIMO applications and additional assets as required. The Live Torrent Control could have performed transcoding of assets to the appropriate formats. This function was superseded by code developed by ULANC (see D8.1.2)

3.1.1.1.5 BBC Headend

The BBC Headend was to provide the SuperNode with which to seed the P2P network and also provide web servers in order to serve some assets via HTTP should they be required.

3.1.2 System in Operation

3.1.2.1 Client Interaction

It was anticipated that *NextShare* client should use the Discovery and Metadata feed tree in order to obtain the necessary information for the client to find and retrieve the desired content. Figure 4 shows how a client walks the tree to find the desired content (in this case the BBC One live stream) and begin streaming. Note that this example is simplified and that in practice a client may have performed operations in parallel on different parts of the discovery feed, in effect caching data for the user in order to provide a more fluid user experience.

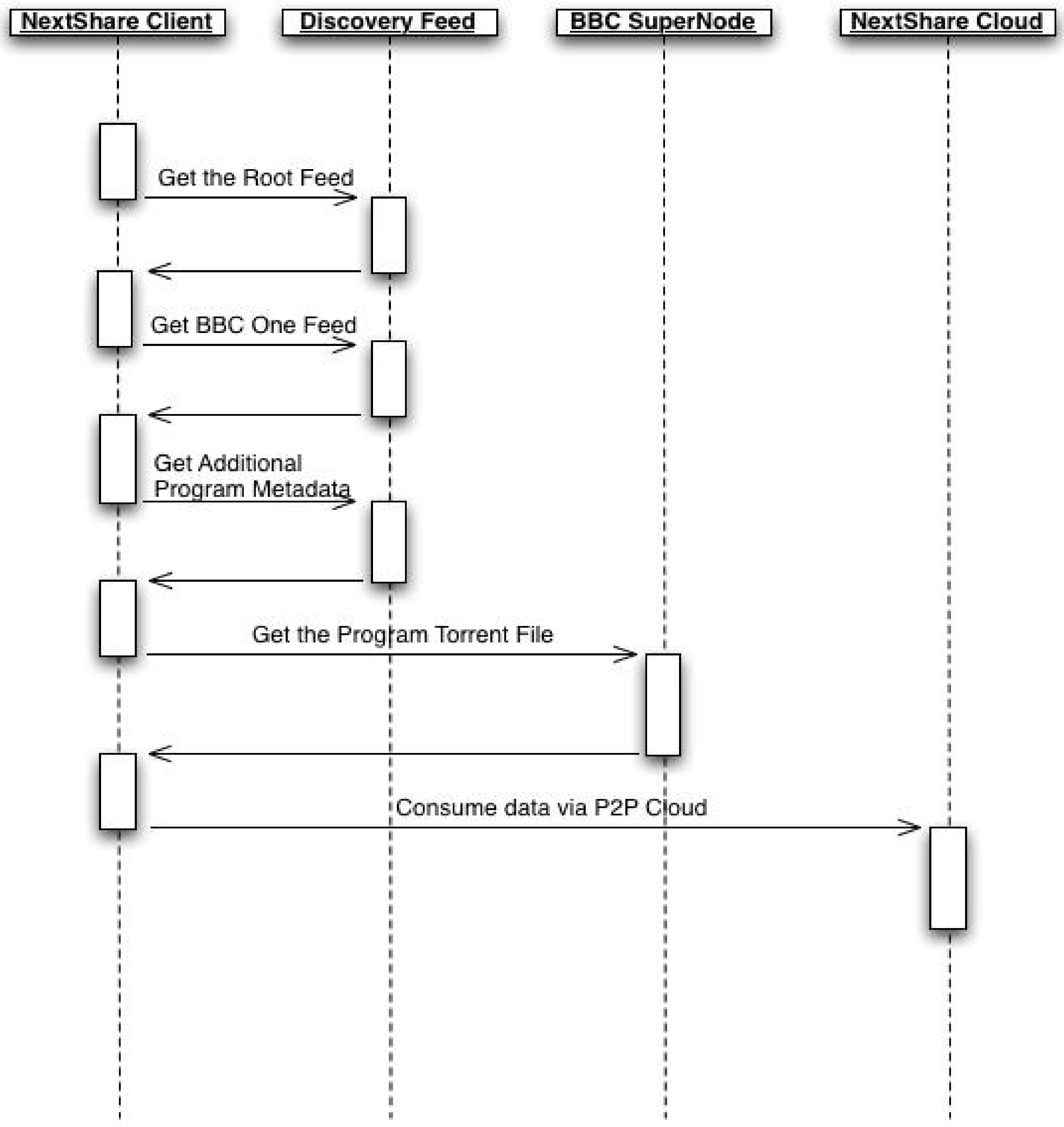


Figure 4: Client interaction example

3.1.2.2 Video on Demand Ingest Example

VOD Ingest was to allow the creation of packages which may contain one or more assets. Packages were to have a window of availability monitored by the VOD Control system. Figure 5 shows the basic interactions to create a package, upload it and then publish it. Once published the VOD Control service would then update the discovery service so the content is discoverable, and when the window closes remove the entry from the discovery service.

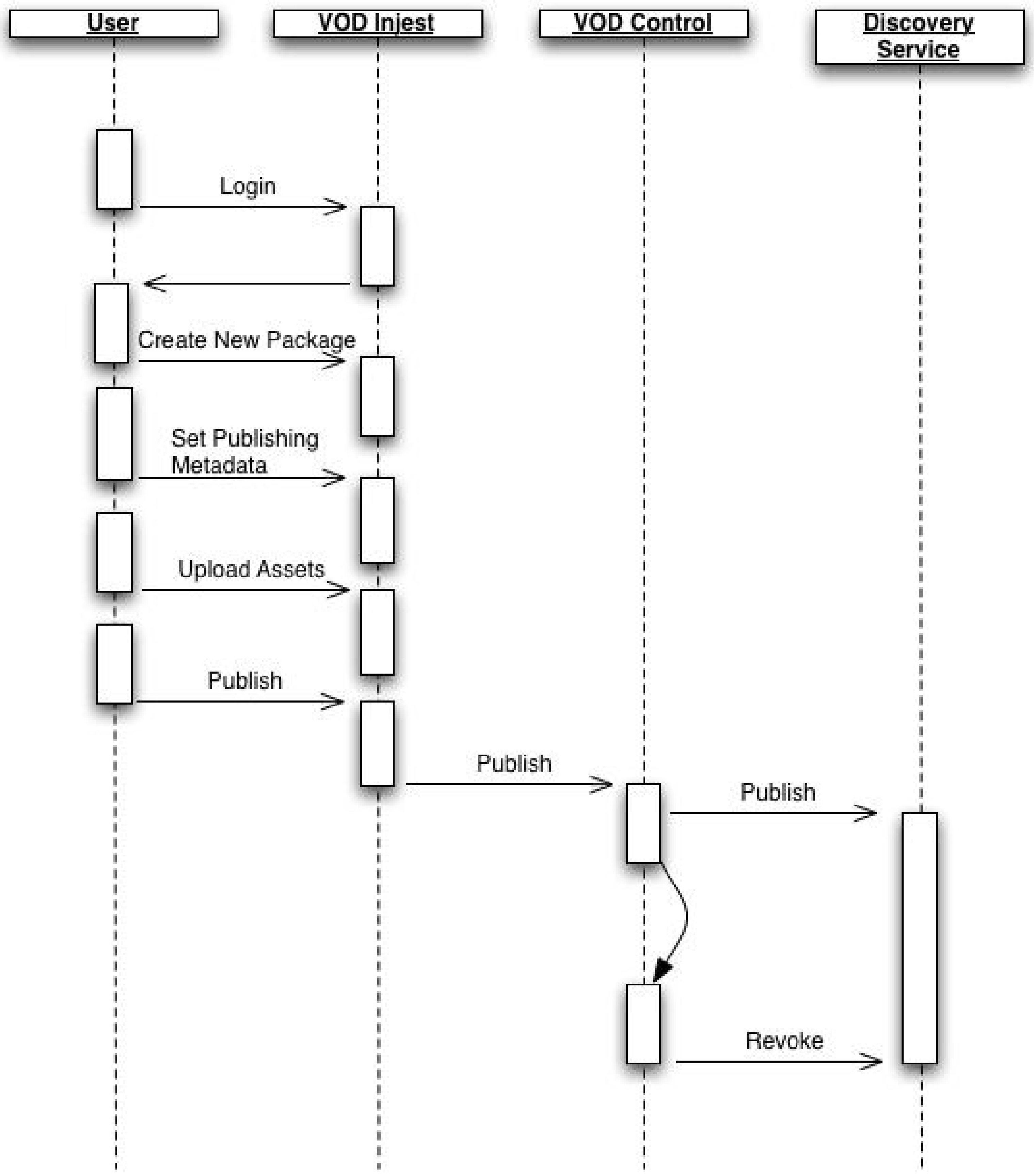


Figure 5: VOD ingest example

3.1.2.3 Ingest Data Model

Figure 6 shows the data model on which the Ingest Platform was based, providing sufficient metadata for the Discovery and Metadata service.

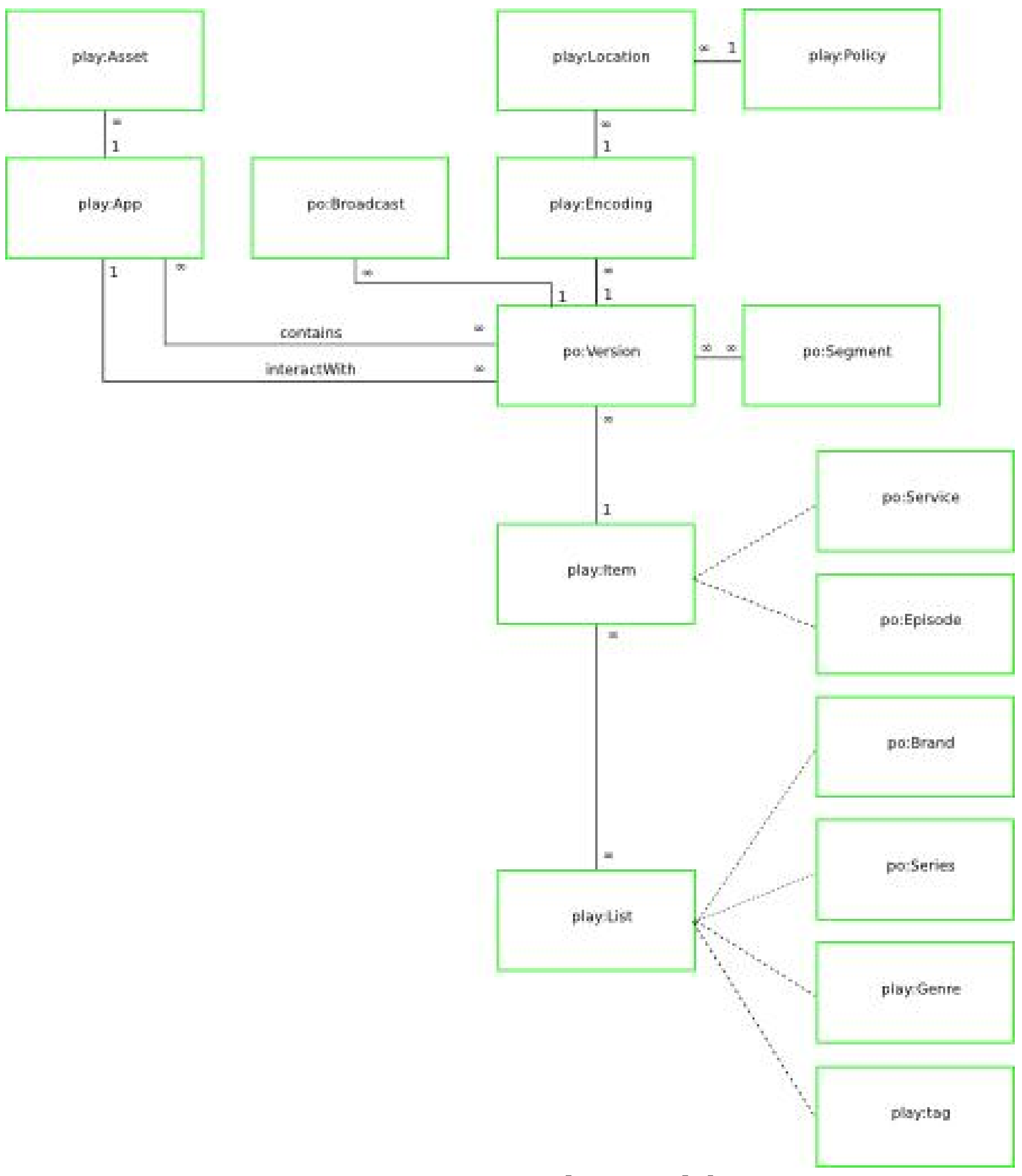


Figure 6: Ingest data model

Overview of the main classes:

•**play:List:** an ordered list of content, typically presented in reverse chronological order.

•**play:Item:** a single content item.

•**po:Version:** a "bag of frames" representing one version of a specific play:Item. Different edits are different versions. A version with sign language embedded is a different version from that without sign language.

•**play:Encoding:** a "bag of bits".

•**play:Location:** a URI.

3.1.3 Alternative solutions for ingest and publishing

3.1.3.1 URIPlay

This solution was devised in 2008 based around the OpenSource URIPlay platform [9]. Whilst URIPlay is a very good metadata aggregation platform, it was decided that implementation would require very specific knowledge of URIPlay and its underlying components, and that a simpler solution which leveraged more of the existing BBC services would be a more practical.

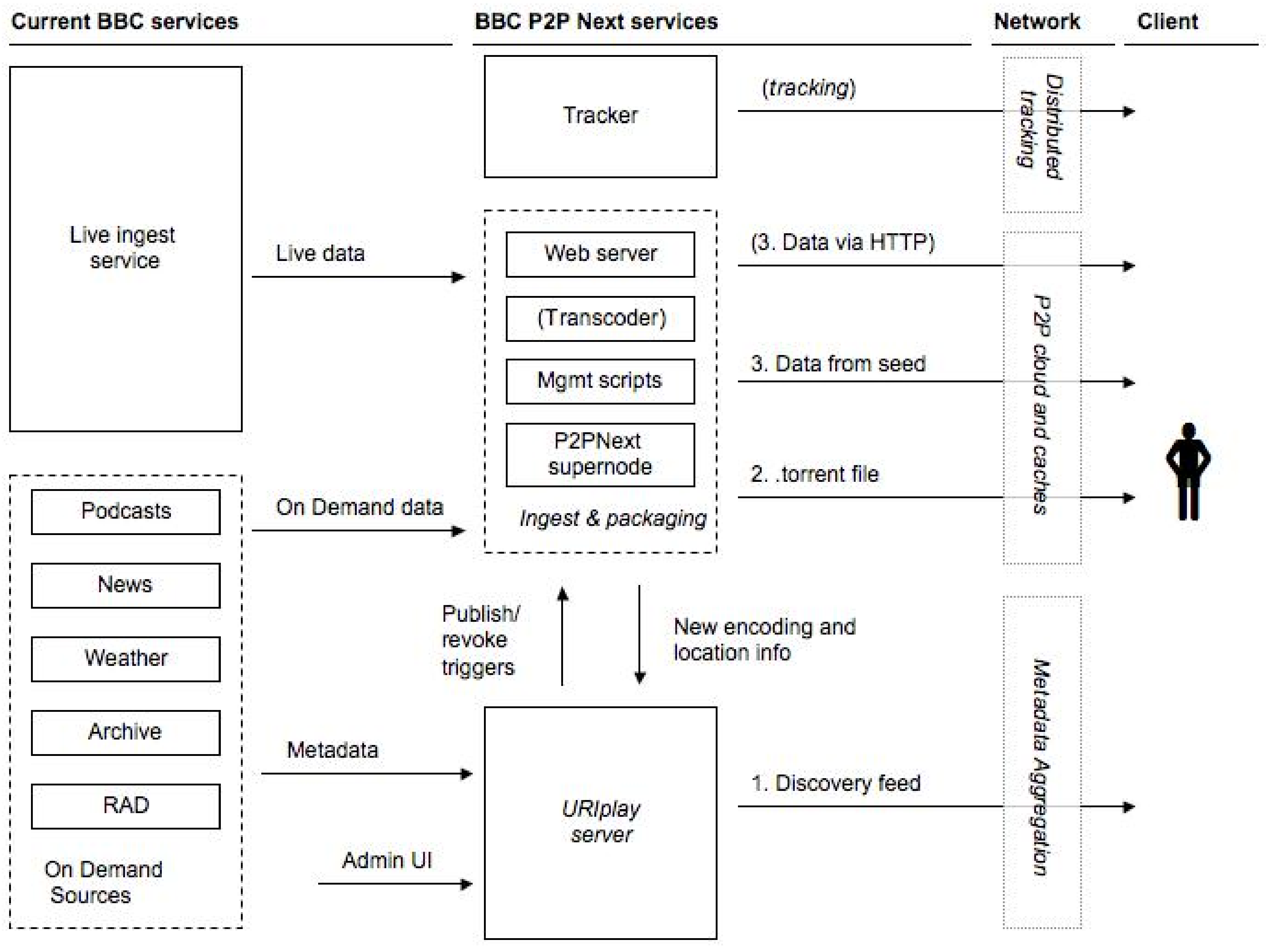


Figure 7: URIPlay Content Ingestion Architecture

3.1.3.2 ULANC implementation for Living Labs Trial

The implementation for the Living Labs Trial were of a very similar nature to that of IBC, however, more channels were seeded and the *NextShare* Swarm was restricted to within the Living Labs network. The P2P Metadata and Discovery feeds remained accessible over the internet.

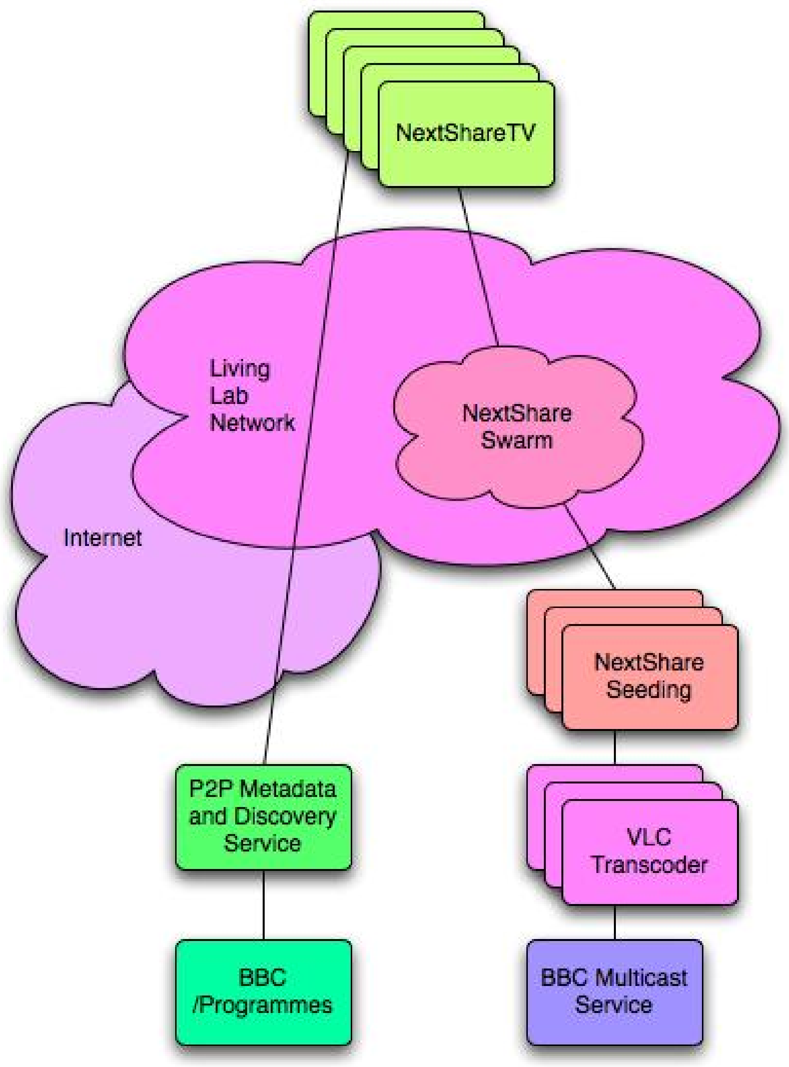


Figure 8: ULANC implementation

3.1.4 Implementation Status

As at M48:

•The discovery and metadata service was implemented in December 2009 and has been running successfully since. It shall continue to be adapted and expanded as required.

•The Live torrent control, VOD torrent control and VOD ingest platform outlined here were replaced with similar functions implemented by ULANC [8].

3.2 JSI content ingest and publishing solution

The content injection solution being developed at JSI tends to cover functionality needed to ingest a professional content provider's content into the *NextShare* system. The basic idea behind the solution is to bridge the content provisioning of content providers with a set of tools enabling the ingestion into a P2P system. However, simple ingestion of the content cannot be the only purpose of the tools. Handling ingestion of the content into a system like *NextShare* requires a broader approach, not only from the professional content provider point of view but from an end user perspective as well [8]. This view, covering broader system tasks, will be called the content provisioning process in the rest of this section. To be able to sketch the scope of the process we will define basic steps that can be performed by the tools in the next section. The steps will be combined into a single scenario of the content provisioning process in Section 3.2.2. The tools design and implementation status is reported in Section 3.2.3. Some conclusions and indications of further work are presented in Section 3.2.4.

3.2.1 Scope of the solution

In this section the basic steps that can be performed by an ingest system are collected. The steps describe elementary functionality required to perform content provisioning tasks. They can be further extended or split in more detailed sub steps at a later time.

3.2.1.1 Content acquisition

Prior to injection in the *NextShare* system the content needs to be acquired. Possible sources of the content are various, like repositories, feeds of any kind, web pages, etc. Most often the acquisition will be related to the current content provider's web technologies based publishing process. The implementation should be uniform and technology independent; its main aim is to acquire content, regardless of its source, format, etc.

3.2.1.2 Store content

The acquired content needs to be stored on the local system prior to the injection of the content into the *NextShare* system. The content can be of various kinds and can include metadata and related content. The type of storage has to be suitable for control and management of the storage as well for further injection into the *NextShare* system.

3.2.1.3 Content adaptation

The content provider needs to adapt the content as obtained from a content source prior to distribution in the system. The adaptation can be related to the content format, packaging, linking to or combining with other content, etc.

3.2.1.4 Content injection

The content provider would like to offer the content to the users of the *NextShare* system but has to prepare the content for injection. Injected content can take two forms: a content unit and a live stream. The types of content are different in the way they are handled in the system. For example, the live content does not require any storage. The content unit intended for provisioning is an independent unit of information suitable for straightforward use by the users. The unit can consist of single or multiple content items, for example files, and has its related metadata. The live stream is content that is constantly delivered. From the content providers perspective, linear broadcast streams should be injected into the *NextShare* system as well. From the end user perspective, web cams and similar devices could be utilized to inject content into the system as live streams. There can be multiple live streams available for injection.

3.2.1.5 Monitor content usage

The content provider would like to track the usage of every content unit injected into the system. Monitoring includes the number and frequency of its usage, distribution paths, geographic distribution of the users, sharing statistics, etc. Usage statistics are available on a regular, predefined basis. The statistics are gathered on one or more system elements in the administration domain of the content provider.

3.2.1.6 Reporting on content distribution and usage

The content provider would like to report on the content distribution. The report should include data like the content's source, the content's form, who, when and how the injection was initiated, where the content was distributed, who has participated in content distribution, etc.

3.2.1.7 Content removal

The content is related to a single content unit and can be distributed through the *NextShare* system. The content provider wants to remove all content from all elements of the system and remove all references to the content, if there are any.

3.2.1.8 Replace content

The old content needs to be removed from any element in the system including replacements of references to the content, if there are any. The old content is replaced with newly published content.

3.2.2 Scenario of system usage through basic steps

The basic steps as presented in the previous section can be combined in a single scenario of system usage related to the content provisioning process as can be seen from the content provider point of view. The scenario is presented as a sequence diagram in figure 9. Besides all the steps described, the scenario needs some additional functionality to be really useful. First, an end user is needed as a receiver of the injected content. He is represented as a user in the diagram. Second, a management functionality needs to be presented in the system which triggers the content injection. Such role can be performed either by a manager or a software component. The role is presented by the manage step in the diagram. Finally, a content distribution takes care that the content is delivered to the end user. This step is being developed in the context of the project's WP4, as this does not belong to the basic content injection steps.
On the top of the figure are labels, representing the basic steps of the process. The label names are abbreviation of a step, for example Acquisition for Content acquisition (3.2.1.1), Store for Store content (3.2.1.2), etc.

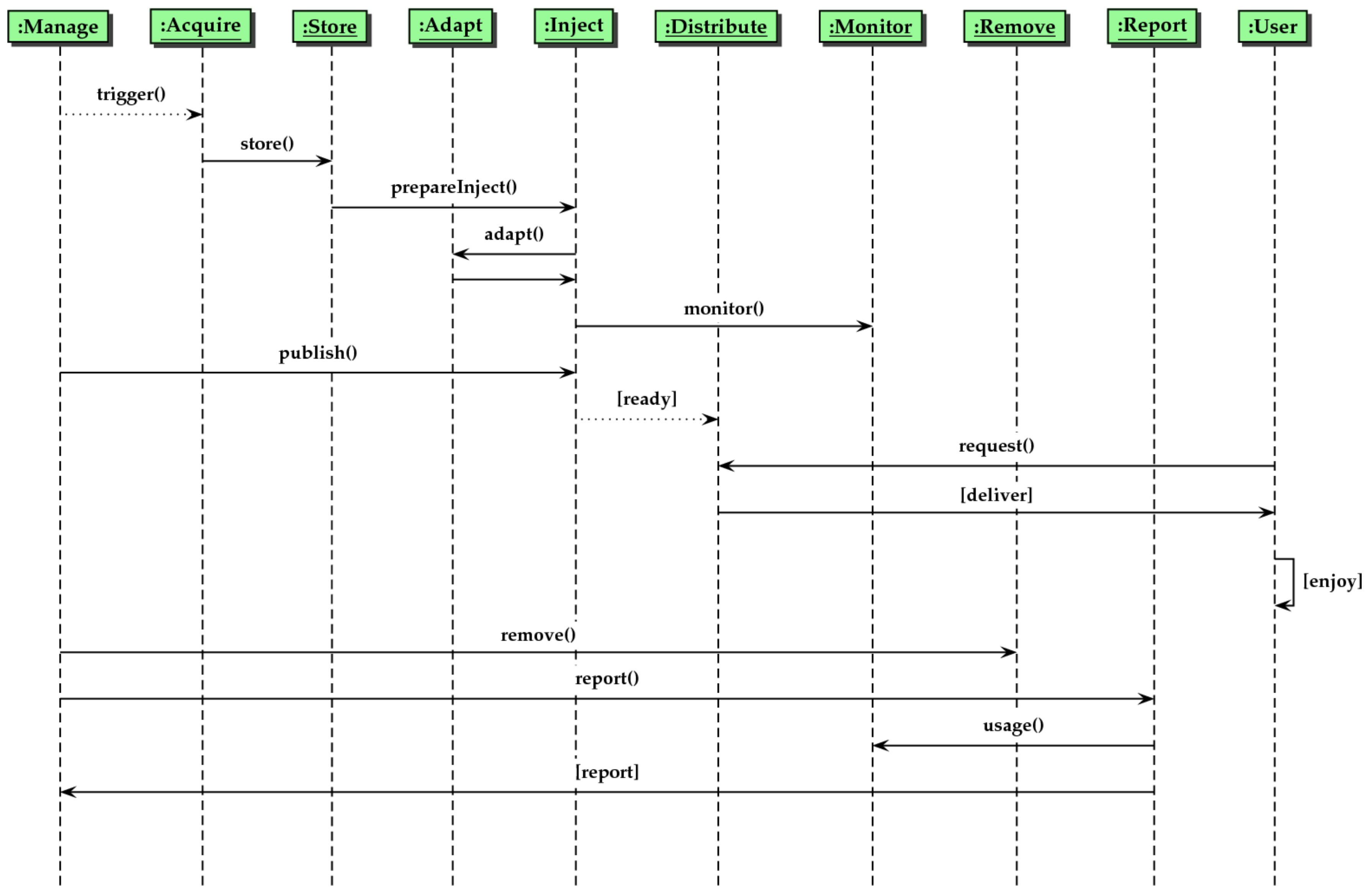


Figure 9: Scenario of the content provisioning process

The scenario starts with triggering content acquisition step (3.2.1.1). The trigger can be manual or automatic. For this reason the line marking the action is dashed. When the content is acquired it is stored (3.2.1.2) on the local system. Later, in the figure presumed automatically, the content is prepared for injection. Depending on the type of target injection the content can be adapted using the content adaptation use case (3.2.1.3). As is indicated in the figure, the adaptation can be triggered automatically if the adaptation is required for the type of injection. All actions so far and after are monitored via the monitoring content usage step (3.2.1.5). When the content is ready for injection it is published through the content injection step, either on demand or live (3.2.1.4). The content distribution state is set to ready. When the user accesses the content the content distribution process starts and the content is delivered to the users. After the content has been consumed and and the request ends, the delivery process stops (from an end user perspective). After the content has been used for some time the content provider can decide to remove the content. The content removal step (3.2.1.7) is used for this purpose. In similar way this step could be replaced or extended with the replace content step (3.2.1.8). The content provider can build a report through reporting on content distribution and usage step (3.2.1.6), based on information gathered by monitoring step. The report can subsequently be used for analysis of the content provisioning process or just for management purposes.

3.2.3 Content provisioning process design

The content provisioning process was designed from the start as a flexible, content provider oriented tool for acquisition and publishing of the content. Though being content provider oriented, it should be usable for ordinary users as well, for publishing their own long tail content. The content provisioning process design is presented in the Figure 10. It shows basic design concepts and their interrelations. The design is split in four parts, content, software, hardware/OS and monitor.

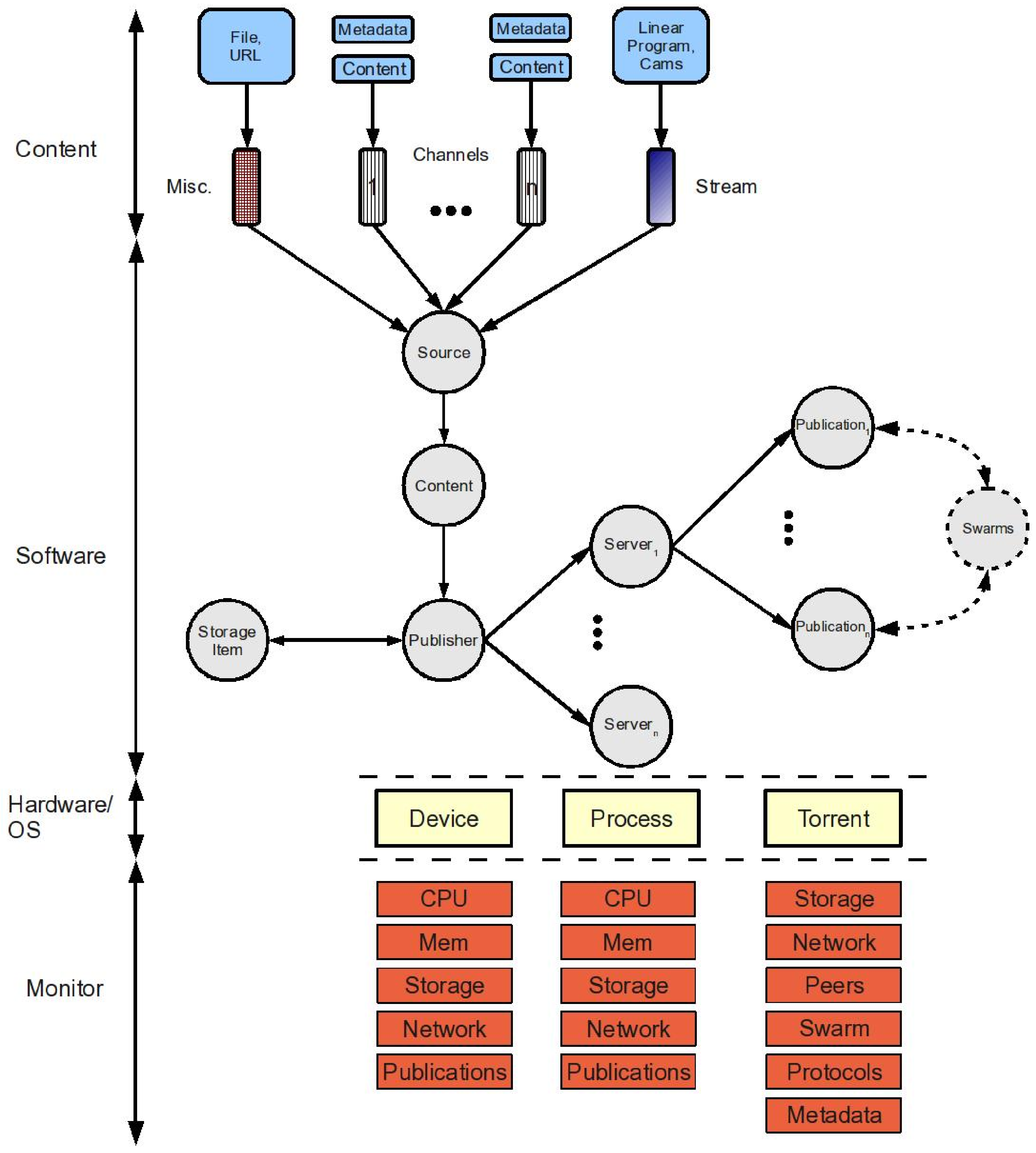


Figure 10: Content provisioning process design

The content part includes currently foreseen content sources. The core source of content are channels that provide one more more items of content. An example of a channel is a RSS feed. The feeds need to be read regularly so new content items can be fetched and made available on the node.

Channels can provide both content essence and metadata. The metadata should be obtained and stored on the node together with the essence. The content can be obtained from other sources as well, like from local file, web location, linear broadcast or locally attached devices like a web cam. The metadata of such content items can be obtained either automatically with the essence processing or can be provided manually.

The software part presents basic software concepts used to implement the process. The concepts are content sources, content items, storage items, publisher, server and publication. The content sources model the content abstractions as discussed in the previous paragraph. They are producers of content items. A content item can be a content essence and/or content metadata. The item stores all content related information.

Content items are stored locally on the node as storage items. The storage item is a concept that handles all content item storage functionality. An example of the storage item is a directory that holds the content essence, related metadata and other software artifacts needed for implementation operations. The storage item then provides all needed functions for storing, deleting, moving or archiving of the content items. Beside an implementation based on a filesystem the process implementation could support storage items implemented on top of a database as well.

The publisher is a core concept of the content provisioning process. It provides needed functionality to guide the entire process, from acquiring the content, storing it and publishing it through one or more servers. It starts, controls and manages various types of servers. An example of a server is a torrent server which enables seeding of the content and keeping track of the content distribution through a local node tracker. When the content item is published through the publisher it becomes a publication. The publication exports needed functionality to manage publishing aspects of one or more content items. The publisher can be related to the “Manage” concept as presented in the scenario of content provisioning process in the figure 9. It provides needed interfaces to control the process and functionality to trigger and guide the process automatically.

The hardware part represents hardware and operating system abstractions and resources used to implement the toolbox. These abstractions are used as a basis for the monitoring part and relate similar concepts together. During the publication process the publisher could use multiple implementation devices with their own resources, like CPU, storage, network, memory, etc. Each device can manage its own set of publications. Processes running on a device really implement the process, using its own share of device resources. In the end publications are related to a single torrent file which hides all content distribution related resources that need to be monitored, like storage, swarm statistics, protocols used, metadata used and exchanged, if any, etc.

3.2.4 Content provisioning process implementation

The implementation has been significantly changed and improved since the last reporting period. The main issues addressed were related to integration of the rich metadata specification in the publishing process, P2P-Next compliant feeds generation (VoD, Live and discovery feeds) and simplification of the existing content ingest implementation. The implementation, named ProviderToolbox, was quite complex and a number of concurrency issues were foreseen for further integration with metadata and feed generation implementations. For this reason the ProviderToolbox implementation was split in several, more manageable components. To ease the integration of rich metadata and to provide rich programmable API interfaces for metadata manipulation a Python based RichMetadata implementation was developed. Existing programmable tools were adapted and extended for provisioning of P2P-Next compliant feeds.

The described tools will be further detailed in next two sections, the first addressing the RichMetadata Python API implementation and the second presenting details on the ProviderToolbox implementation. The sections provide some insights on the implementation of the tools, their usage and performance measurements.

3.2.5 RichMetadata Python implementation

This description is a shortened, but additionally annotated version of the tool documentation. The implementation has the following features:

- parses core, advertising, payment and scalability metadata set
- build core, advertising, payment and scalability metadata set
- supports TVA and Mpeg7 metadata types
- parses and builds MPEG 21 DID metadata
- provides metagen tool to generate supported metadata
- provides dynamic RichMetadata APIs according to the metadata set for programmable manipulation
- integrates core metadata set with the Next-Share core via torrent creation and show tool
- supports both ElementTree and cElementTree. Since results are better, of course, using the native C library is the default choice for the implementation

The implementation has the following drawbacks:

- it has very naive XML parsing/building implementation, based on the python ElementTree - but basic mechanisms are there so it is easily extensible. The consequence is that the tool can generate only as much metadata as has been learned from samples.
- doesn't support URIPlay metadata

The implementation is based on the python native ElementTree module. It is centered around P2P-Next defined tags and TVA tags, as defined in conf/RichMetadataSettings.py, that map to single RichMetadata class for any other format supported (like MPEG7). The implementation then allows mapping between various formats.

The core interface for the implementation is the RichMetadataGenerator: a singleton class that learns during initialization from the supplied XML samples in the conf directory. Since the implementation depends on samples be careful when modifying them.

The RichMetadataGenerator provides the following methods:

- RichMetadataGenerator.getInstance: returns the RichMetadataGenerator singleton instance
- RichMetadataGenerator.getRichMetadata: method is a factory for RichMetadata instances, either from scratch or from input source
- RichMetadataGenerator.build: builds target metadata XML representation
- RichMetadataGenerator.prettyPrint: prints XML representation in human readable fashion - for debugging purposes only

The methods, parameters, and results are further describe via standard Python documentation in the code.

The RichMetadata instance provides its API dynamically, based on the type of metadata set, learning and settings. Please note that since the methods are provided dynamically they depend on the samples provided and the settings in RichMetadataSettings.py.

Since the getters and setters names provided are straightforward no other documentation of the API is provided. If in doubt while programming consult RichMetadata instance metadataType and use getAPIMethods to get the list of API methods of the current instance. For scalability metadata attributes SPS and PPS consult the metagen tool help. An examples of usage of the dynamic APIs is provided in RichMetadataTest.py.

3.2.5.1 Example of usage

3.2.5.1.1 The metagen tool

First, export the PYTHONPATH:

```
xyz:~/src/Next-Share:{1}> export PYTHONPATH=$(pwd):.
```

Use the metagen.py tool to generate XML metadata, consult the tool help for various options:

```
xyz:~/src/Next-Share:{1}> python JSI/RichMetadata/tools/metagen.py -h
```

In the following, command line examples are provided. The copied examples can be used to generate desired output if the Python settings are set and the tools installed properly.

3.2.5.1.1.1 Generating core metadata

The metagen tool can generate core metadata of type TVA and MPEG7. For example, the following command will generate core metadata of type TVA:

```
python JSI/RichMetadata/tools/metagen.py --aspectRatio=4:3 --audioCoding="MPEG-1 Audio Layer
III" --bitRate=524288 --captionLanguage=English --copyrightNotice="Copyright 2010 BBC"
--duration=P0Y0M0DT0H0M35S --entityIdentifier="1" --fileFormat=mp4 --fileSize=83191350
--frameRate=30 --genre=Infotainment --horizontalSize=640 --howRelated="Content Package"
--language=English --mediaLocator="http://p2p-next.org/images/example1.jpg" --minimumAge=12
--numOfChannels=2 --originator=p2p-next --productionDate="2008-07-10T10:00:00+01:00"
--productionLocation=AT --programId="crid://p2p-next/example1" --publisher=p2p-next
--relatedMaterial="related" --releaseDate=2008-07-14 --signLanguage=English --synopsis="John
and Jane Doe visit a car dealer to buy a new car for John" --titleEpisodeTitle="John Doe buys
a car" --titleMain="Next-Share Test Stream 1" --titleSeriesTitle="The Doe Family"
--verticalSize=480 --videoCoding="MPEG-2 Video Main Profile @ Main Level" --region=UK
--content=audiovisual
```

Corresponding MPEG7 metadata could be generated by adding an "-f Mpeg7" option to the previous command line. The result should be, after running the command, an MPEG7 compliant metadata.

3.2.5.1.1.2 Generation of optional metadata

Optional metadata could be created for the advertising and payments, for example the following command will generate advertising metadata of type TVA, where the "-a" option specifies the metadata type.

```
xyz:~/src/Next-Share:{1}> python JSI/RichMetadata/tools/metagen.py -a --businessModel=BM2
--isLiveContent=false --allowAdvertising=true --circularContent="Allow Superdistribution"
--adType="Web" --streamingType="In-stream" --adFormat="banner" --age=25 --gender=F
--country=SI --aspectRatio="16:9" --verticalSize=405 --horizontalSize=720 --frameRate=30
--publisher="p2p-next"
```

Similarly, payment metadata is created using the "-p" option:

```
xyz:~/src/Next-Share:{1}> python JSI/RichMetadata/tools/metagen.py -p --acceptDonations=true
--advancedInfos="http://www.p2p-next.org/paymentInformation.html" --currency=EUR
--paymentId=xyz --paymentRecipient=me --price=1500 --publisher="JSI"
```

Invoking the tool with the following command will generate scalability data as specified in the samples. Please note the "-s" switch. For specifying the SPS and PPS values, please consult the metagen tool help.

```
xyz:~/src/Next-Share:{1}> python JSI/RichMetadata/tools/metagen.py -s
--adaptationOperatorDependencyId="0 1 2 3" --adaptationOperatorQualityLevel="0 0 0 0"
--utilityFramerate="25 25 25 25" --utilityHeight="240 240 480 480" --utilityWidth="320 320
640 640" --constraintBitrate="400 800 1200 240" --adaptationOperatorTemporalLevel="0 0 0 0"
--pPS="spsId=1,value=23:spsId=2,value=24"
--sPS="spsId=1,width=a,height=b,value=cde:spsId=2,width=a,height=b,value=cde"
```

3.2.5.1.1.3 Generation of MPEG 21 DID metadata

The metagen tool can generate the MPEG 21 DID metadata. In the following two examples the first invocation of the tool generates the core metadata for type TVA in a compact form ("-c" option) and the command redirects the standard output to file core-meta.xml. The second invocation of the tool the generated file is included into the DID with the option "–metaCore=core-meta.xml". The second command demonstrates the options of the DID base document generation as well. Please note that the DID base document gets generated if the option "-b" is provided. The second command generates the compact version of the XML ("-c") and redirects the output to did-base.xml. This file is now ready to be included in the torrent file. An example of this will be presented in the next section.

```
xyz:~/src/Next-Share:{1}> python JSI/RichMetadata/tools/metagen.py --aspectRatio=16:9
--audioCoding="MPEG-1 Audio Layer III" --bitRate=80000 --captionLanguage=si
--duration=P0Y0M15DT2H1M12S --fileFormat=mp4 --fileSize=12345432 --frameRate=30
--genre=Codatainment --horizontalSize=720 --language=si --minimumAge=3 --numOfChannels=2
--originator=JSI --productionDate=2010-08-16 --productionLocation=SI --publisher=p2p-next
--releaseDate=2010-08-17 --signLanguage=si --synopsis="Fine metadata tools"
--titleEpisodeTitle="RichMetadata v0.1" --titleMain="P2P-Next code"
--titleSeriesTitle="RichMetadata tools" --verticalSize=405 --videoCoding="MPEG-2 Video Main
Profile @ Main Level" -c > core-meta.xml

xyz:~/src/Next-Share:{1}> python JSI/RichMetadata/tools/metagen.py -b --identifier="urn:p2p-
next:item:rtv-slo-slo1-xyz" --relatedIdentifier="urn:rtv-slo:slo1-xyz"
--contentType="video/ts" --contentReference="URI to video included in the torrent"
--advertisementReference="URI to additional MPEG_21 data (advertising)" --limoReference="URI
to additional MPEG_21 data (limo)" --paymentReference="URI to additional MPEG_21 data
(payment)" --scalabilityReference="URI to additional MPEG_21 data (scalability)"
--metaCore=core-meta.xml -c > did-base.xml
```

Additional MPEG 21 DID metadata, the metadata that is distributed in the torrent itself, can be generated in similar fashion as the base metadata. Since it can include multiple files first the files need to be provided or generated. Metadata file such as payments, advertising and scalability can be generated as presented in the previous section while redirecting the output of the command to a properly named file.

Additional files, like logo and limo files (html, javascript and css) can to be provided as well. For the purpose of this example we will generate them on the fly, as simple strings:

```
xyz:~/src/Next-Share:{1}> echo "Limo CSS content, should be included as XML CDATA" > limo.css
xyz:~/src/Next-Share:{1}> echo "Limo HTML content, should be included as XML CDATA" >
limo.html
xyz:~/src/Next-Share:{1}> echo "Limo Javascript content, should be included as XML CDATA" >
limo.js
xyz:~/src/Next-Share:{1}> echo "Logo graphics" > logo.png
```

When all files are available in proper format the additional MPEG 21 DID metadata can be generated by running the command as presented in the next example:

```
xyz:~/src/Next-Share:{1}> python JSI/RichMetadata/tools/metagen.py -d
--relatedIdentifier="urn:rtv-slo:slo1-xyz" --cSSName="Name of the CSS file in Limo"
--css=limo.css --html=limo.html --javascript=limo.js --javascriptName="Name of Javascript
file in Limo" --logo=logo.png --logoName="Name of the logo file" --logoReference="URI
reference to the logo (Needed indeed?)" --logoType="image/png"
--metaAdvertisement=advertising-meta.xml --metaPayment=payment-meta.xml
--metaScalability=scalability-meta.xml -c > did-additional.xml
```

It should be noted that the options that are related to files expect the files to be provided. If not the tool will exit with a meaningful message and help output.

3.2.5.1.2 Programming interface

The same examples as presented in the previous section for the metagen tool can be easily programmed as well. To get some insights and inspiration how the RichMetadata APIs can be used, see RichMetadataTest.py. For a start some hints are provided for a python run from the command line, please not the additional comments (#) in the example text:

```
  xyz:~/src/Next-Share:{1}> export PYTHONPATH=$(pwd):.
  xyz:~/Next-Share:{746}> python
  Python 2.6.5 (r265:79063, Apr 16 2010, 13:09:56)
  [GCC 4.4.3] on linux2
  Type "help", "copyright", "credits" or "license" for more information.
  >>> from JSI.RichMetadata.RichMetadata import RichMetadataGenerator
  >>> rmg = RichMetadataGenerator.getInstance()
  # Default metadata type is core
  >>> rm = rmg.getRichMetadata()
  >>> rm.getAPIMethods() # Lists all instance methods
  ['getAspectRatio', 'getAudioCoding', 'getBitRate', 'getCaptionLanguage',
  'getContent', 'getCopyrightNotice', 'getDuration', 'getEntityIdentifier',
  'getFileFormat', 'getFileSize', 'getFrameRate', 'getGenre',
  'getHorizontalSize', 'getHowRelated', 'getLanguage', 'getMediaLocator',
  'getMinimumAge', 'getNumOfChannels', 'getOriginator', 'getProductionDate',
  'getProductionLocation', 'getProgramId', 'getPublisher', 'getRegion',
  'getReleaseDate', 'getSignLanguage', 'getSynopsis', 'getTitleEpisodeTitle',
  'getTitleMain', 'getTitleSeriesTitle', 'getVerticalSize', 'getVideoCoding',
  'setAspectRatio', 'setAudioCoding', 'setBitRate', 'setCaptionLanguage',
  'setContent', 'setCopyrightNotice', 'setDuration', 'setEntityIdentifier',
  'setFileFormat', 'setFileSize', 'setFrameRate', 'setGenre',
  'setHorizontalSize', 'setHowRelated', 'setLanguage', 'setMediaLocator',
  'setMinimumAge', 'setNumOfChannels', 'setOriginator', 'setProductionDate',
  'setProductionLocation', 'setProgramId', 'setPublisher', 'setRegion',
  'setReleaseDate', 'setSignLanguage', 'setSynopsis', 'setTitleEpisodeTitle',
  'setTitleMain', 'setTitleSeriesTitle', 'setVerticalSize', 'setVideoCoding']
  >>> rm.setTitleMain("My main title")
  # setters return instance of metadata, avoided in next example calls
  <JSI.RichMetadata.RichMetadata.RichMetadata object at 0xb773e5ec>
  >>> rm.setTitleSeriesTitle("My series title")
  >>> rm.setTitleEpisodeTitle("Experiment with rich metadata")
  # The supplied values should be all strings
  >>> rm.setFrameRate("30").setBitRate("203010")
  >>> rm.setHorizontalSize("640").setVerticalSize("480").setAspectRatio("4:3")
  # Default presentation is TVA
  >>> rm_xml_tva = rmg.build(rm)
  # Load the settings to get access to proper variables
  >>> from JSI.RichMetadata.conf import metadata
  # Request MPEG7 representation of the same metadata
  >>> rm_xml_mpeg7 = rmg.build(rm, metadata.TAG_MPEG7)
  # Print the TVA representation to stdout
  >>> print rmg.prettyPrint(rm_xml_tva)
  <?xml version="1.0" ?>
  <TVAMain xmlns="urn:tva:metadata:2007" xmlns:mpeg7="urn:mpeg:mpeg7:schema:2001
" xmlns:mpeg7_tva="urn:tva:mpeg7:2005" xmlns:p2pnext="urn:p2pnext:metadata:2008"
xmlns:tva="urn:tva:metadata:2007" xmlns:xsi="http://www.w3.org/2001/XMLSchema-i
nstance" xsi:schemaLocation="urn:tva:metadata:2007 tva_metadata_3-1_v141_p2p.xsd
">
     <ProgramDescription>
        <ProgramInformationTable>
           <ProgramInformation>
              <BasicDescription type="p2pnext:BasicP2PDataDescriptionType">
                 <Title type="main">
                    My main title
                 </Title>
                 <Title type="seriesTitle">
                    My series title
                 </Title>
                 <Title type="episodeTitle">
                    Experiment with rich metadata
```

```
                  </Title>
               </BasicDescription>
               <AVAttributes>
                  <BitRate>
                     203010
                  </BitRate>
                  <VideoAttributes>
                     <HorizontalSize>
                        640
                     </HorizontalSize>
                     <VerticalSize>
                        480
                     </VerticalSize>
                     <AspectRatio>
                        4:3
                     </AspectRatio>
                     <FrameRate>
                        30
                     </FrameRate>
                  </VideoAttributes>
               </AVAttributes>
            </ProgramInformation>
         </ProgramInformationTable>
      </ProgramDescription>
   </TVAMain>
   # And similar example for payments data
   # And similar example for payments data
   >>> rmp = rmg.getRichMetadata(None, metadata.METADATA_PAYMENT)
   >>> rmp.getAPIMethods()
   ['getAcceptDonations', 'getAdvancedInfos', 'getCurrency', 'getPaymentId',
   'getPaymentRecipient', 'getPrice', 'getProgramId', 'getPublisher',
   'setAcceptDonations', 'setAdvancedInfos', 'setCurrency', 'setPaymentId',
   'setPaymentRecipient', 'setPrice', 'setProgramId', 'setPublisher']
   >>> rmp.setAcceptDonations("True")
   >>> rmp.setCurrency("Euro")
   >>> rmp.setPaymentRecipient("me@home")
   >>> rmp.setPrice("A lot")
   >>> rmp_xml_mpeg7 = rmg.build(rmp, metadata.TAG_MPEG7)
   >>> print rmg.prettyPrint(rmp_xml_mpeg7)
   <?xml version="1.0" ?>
   <Mpeg7 xmlns="urn:mpeg:mpeg7:schema:2001" xmlns:mpeg7="urn:mpeg:mpeg7:schema:2001"
 xmlns:p2pnext="urn:p2pnext:metadata:2008" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
 instance" xsi:schemaLocation="urn:mpeg:mpeg7:schema:2001 mpeg7-v1_p2p.xsd">
      <Description type="p2pnext:P2POptionalDescriptionType">
         <p2pnext:PaymentData type="p2pnext:P2PPaymentDataType">
            <p2pnext:Price currency="Euro">
               A lot
            </p2pnext:Price>
            <p2pnext:PaymentRecipient>
               me@home
            </p2pnext:PaymentRecipient>
            <p2pnext:AcceptDonations>
               True
            </p2pnext:AcceptDonations>
         </p2pnext:PaymentData>
      </Description>
   </Mpeg7>
```

3.2.5.1.3 Integration in the core

The RichMetadata prototype implementation provides a modified createtorrent.py and btshowmetainfo.py. The modifications are minimal to allow to include the metadata into the torrent file via methods provided in the Next-Share core API.

First, appropriate metadata is generated using the metagen tool. During generation the compact option (-c) provides a compact representation of the metadata for inclusion into torrent file. The commands suitable for this purpose were presented in Section 3.2.5.1.1.3 in the case of generating base DID data.

The torrent file can subsequently be created as follows:

```
xyz:~/src/Next-Share:{1}> python JSI/RichMetadata/tools/createtorrent.py
--source README.txt --meta did-base.xml
```

The torrent file now includes the base DID metadata.

After running the btshowmetainfo.py its output should show the output similar to the following, some debugging information omitted:

```
xyz:~/src/Next-Share:{1}> python JSI/RichMetadata/tools/btshowmetainfo.py README.txt.tstream
...
Publisher = Tribler
Description =
Progressive = 1
Title = README.txt
Creation Date = 1282075652
Content Hash = PT3GQCPW4NPT6WRKKT25IQD4MU5HM4UY
Speed Bps = 0.688611090183
Revision Date = 1282075652
metainfo file.: README.txt.tstream
info hash.....: 729c70cf6d03749ea89aec1b07a6bf903b305099
info hash.....: 'r\x9cp\xcfm\x03t\x9e\xa8\x9a\xec\x1b\x07\xa6\xbf\x90;0P\x99'
file name.....: README.txt
file size.....: 2479 (0 * 32768 + 2479)
announce url..: http://127.0.0.1:6969/announce
ns-metadata...:
Publisher = Tribler
Description =
Progressive = 1
Title = README.txt
Creation Date = 1282918421
Content Hash = PT3GQCPW4NPT6WRKKT25IQD4MU5HM4UY
Speed Bps = 0.688611090183
Revision Date = 1282918421
metainfo file.: README.txt.tstream
info hash.....: 642d0dccd811a60505dfa9cce8687e074739da22
info hash.....: 'd-\r\xcc\xd8\x11\xa6\x05\x05\xdf\xa9\xcc\xe8h~\x07G9\xda"'
file name.....: README.txt
file size.....: 2479 (0 * 32768 + 2479)
announce url..: http://127.0.0.1:6969/announce
ns-metadata...:
<?xml version="1.0" ?>
<DIDL xmlns:didl="urn:mpeg:mpeg21:2002:02-DIDL-NS" xmlns:dii="urn:mpeg:mpeg21:2002:01-DII-NS"
xmlns:xi="http://www.w3.org/2001/XInclude" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="urn:mpeg:mpeg21:2002:02-DIDL-NS didl.xsd
urn:mpeg:mpeg21:2002:01-DII-NS dii.xsd">
   <Item>
      <Descriptor>
         <Statement mimeType="text/xml">
            <dii:Identifier>
               urn:p2p-next:item:rtv-slo-slo1-xyz
            </dii:Identifier>
         </Statement>
      </Descriptor>
      <Descriptor>
         <Statement mimeType="text/xml">
            <dii:RelatedIdentifier relationshipType="urn:mpeg:mpeg21:2002:01-RDD-
NS:IsAbstractionOf">
               urn:rtv-slo:slo1-xyz
            </dii:RelatedIdentifier>
         </Statement>
      </Descriptor>
      <Descriptor>
         <Statement mimeType="text/xml">
            <dii:Type>
               urn:p2p-next:type:item:2009
            </dii:Type>
         </Statement>
      </Descriptor>
      <Descriptor>
         <Descriptor>
            <Statement mimeType="text/xml">
               <dii:Type>
                  urn:p2p-next:type:rm:core:2009
               </dii:Type>
            </Statement>
         </Descriptor>
         <xi:include href="URI to additional MPEG_21 data (payment)" xpointer="rm.payment"/>
         <xi:include href="URI to additional MPEG_21 data (advertising)"
xpointer="rm.advertisement"/>
         <xi:include href="URI to additional MPEG_21 data (scalability)"
xpointer="rm.scalability"/>
         <Statement mimeType="text/xml">
            &lt;TVAMain publisher=&quot;p2p-next&quot;
xmlns=&quot;urn:tva:metadata:2007&quot; xmlns:mpeg7=&quot;urn:mpeg:mpeg7:schema:2001&quot;
xmlns:mpeg7_tva=&quot;urn:tva:mpeg7:2005&quot;
xmlns:p2pnext=&quot;urn:p2pnext:metadata:2008&quot;
xmlns:tva=&quot;urn:tva:metadata:2007&quot; xmlns:xsi=&quot;http://www.w3.org/2001/XMLSchema-
instance&quot; xsi:schemaLocation=&quot;urn:tva:metadata:2007 tva_metadata_3-
1_v141_p2p.xsd&quot;&gt;&lt;ProgramDescription&gt;&lt;ProgramInformationTable&gt;&lt;ProgramI
nformation&gt;&lt;BasicDescription
type=&quot;p2pnext:BasicP2PDataDescriptionType&quot;&gt;&lt;Title
type=&quot;main&quot;&gt;P2P-Next code&lt;/Title&gt;&lt;Title
type=&quot;seriesTitle&quot;&gt;RichMetadata tools&lt;/Title&gt;&lt;Title
type=&quot;episodeTitle&quot;&gt;RichMetadata v0.1&lt;/Title&gt;&lt;Synopsis&gt;Fine metadata
tools&lt;/Synopsis&gt;&lt;Genre
href=&quot;urn:mpeg:mpeg7:cs:GenreCS:2001&quot;&gt;&lt;Name&gt;Codatainment&lt;/Name&gt;&lt;/
Genre&gt;&lt;ParentalGuidance&gt;&lt;mpeg7_tva:MinimumAge&gt;3&lt;/mpeg7_tva:MinimumAge&gt;&l
t;/ParentalGuidance&gt;&lt;Language&gt;si&lt;/Language&gt;&lt;CaptionLanguage&gt;si&lt;/Capti
onLanguage&gt;&lt;SignLanguage&gt;si&lt;/SignLanguage&gt;&lt;ProductionDate&gt;&lt;TimePoint&
gt;2010-08-
16&lt;/TimePoint&gt;&lt;/ProductionDate&gt;&lt;ProductionLocation&gt;SI&lt;/ProductionLocatio
n&gt;&lt;ReleaseInformation&gt;&lt;ReleaseDate&gt;&lt;DayAndYear&gt;2010-08-
17&lt;/DayAndYear&gt;&lt;/ReleaseDate&gt;&lt;/ReleaseInformation&gt;&lt;Duration&gt;P0Y0M15DT
2H1M12S&lt;/Duration&gt;&lt;p2pnext:Originator&gt;JSI&lt;/p2pnext:Originator&gt;&lt;/BasicDes
cription&gt;&lt;AVAttributes&gt;&lt;FileFormat
href=&quot;urn:mpeg:mpeg7:cs:FileFormatCS:2001&quot;&gt;&lt;Name&gt;mp4&lt;/Name&gt;&lt;/File
Format&gt;&lt;FileSize&gt;12345432&lt;/FileSize&gt;&lt;BitRate&gt;80000&lt;/BitRate&gt;&lt;Au
dioAttributes&gt;&lt;Coding
href=&quot;urn:mpeg:mpeg7:cs:AudioCodingFormatCS:2001&quot;&gt;&lt;Name&gt;MPEG-1 Audio Layer
III&lt;/Name&gt;&lt;/Coding&gt;&lt;NumOfChannels&gt;2&lt;/NumOfChannels&gt;&lt;/AudioAttribut
es&gt;&lt;VideoAttributes&gt;&lt;Coding
href=&quot;urn:mpeg:mpeg7:cs:VisualCodingFormatCS:2001&quot;&gt;&lt;Name&gt;MPEG-2 Video Main
Profile @ Main
Level&lt;/Name&gt;&lt;/Coding&gt;&lt;HorizontalSize&gt;720&lt;/HorizontalSize&gt;&lt;Vertical
Size&gt;405&lt;/VerticalSize&gt;&lt;AspectRatio&gt;16:9&lt;/AspectRatio&gt;&lt;FrameRate&gt;3
0&lt;/FrameRate&gt;&lt;/VideoAttributes&gt;&lt;/AVAttributes&gt;&lt;/ProgramInformation&gt;&l
t;/ProgramInformationTable&gt;&lt;/ProgramDescription&gt;&lt;/TVAMain&gt;

         </Statement>
      </Descriptor>
      <xi:include href="URI to additional MPEG_21 data (limo)" xpointer="limo"/>
      <Component>
         <Resource mimeType="video/ts" ref="URI to video included in the torrent"/>
      </Component>
   </Item>
</DIDL>
```

3.2.5.2 Implementation performance

The implementation performance can be estimated by running the RichMetadataTest.py on your own machine. For this purpose you need to set the variable "runspeed" to true in RichMetadataTest.py. Most of the parsing and building tests are based on samples provided in the conf directory. The example values, obtained on quite capable machine, run as tests on 10 batches, each with 100 iterations of stated test:

Test	Time value*100+-std.	Oper/s
Parsing XML TVA	0.10501+-1.35%	952.3
Parsing XML MPEG7	0.12632+-0.75%	791.6
Building XML TVA	0.14411+-0.72%	693.9
Building XML MPEG7	0.20205+-0.77%	494.9
Sparse build TVA	0.16697+-0.64%	598.9

Sparse build MPEG7	0.16653+-0.37%	600.5
Cross build TVA	0.3101+-0.6%	322.5
Cross build MPEG7	0.2731+-0.59%	366.2
TVA from scratch	0.18575+-1.0%	538.4
MPEG7 from scratch	0.24513+-0.71%	407.9
Parsing payments TVA	0.03978+-1.15%	2513.7
Parsing advert. TVA	0.06527+-1.46%	1532.2
Parsing scalab. TVA	0.09416+-0.71%	1062.0
Parsing paym. MPEG7	0.033+-2.5%	3030.2
Parsing advert. MPEG7	0.05911+-0.39%	1691.8
Parsing scalab. MPEG7	0.08799+-1.51%	1136.5
Build payments TVA	0.05275+-0.51%	1895.7
Build advert. TVA	0.08734+-1.17%	1145.0
Build scalab. TVA	0.1544+-1.05%	647.7
Build paym. MPEG7	0.03841+-2.54%	2603.5
Build advert. MPEG7	0.08176+-0.76%	1223.1
Build scalab. MPEG7	0.14322+-0.85%	698.2
Parse DID base	0.06628+-2.14%	1508.8
Parse DID addit.	0.09898+-0.29%	1010.3
Build DID base	0.11977+-0.76%	834.9
Build DID addit.	0.15257+-0.49%	655,4

The default implementation that the tools use is cElementTree. Though it is included in the Python core (v2.5 and up) some distributions could choose that the package holding cElementTree is distributed separately. In such case install the missing package or use the ElementTree implemented in pure Python (see the import statement in RichMetadata.py).

3.2.6 Provider Toolbox

The ProviderToolbox provides a set of tools that ease ingest of content into the P2P-Next enabled cloud. In this document the second version of the tool is described. The goal for the second version was to simplify the implementation of the original toolbox.

Currently the toolbox provides the following features:

- can read RSS and Atom feeds
- can generate rich metadata from the feeds and embodied media
- provides programmable tools for generating P2P-Next compliant Atom feeds (VoD, Live streams, discovery):
  - for live streams provides parsers for RTV Slovenia and BBC programme schedule
- can fetch the content from a feed and store it locally
- can generate torrent files for fetched content according to the P2P-Next specification
- can generate P2P-Next VoD feeds from external VoD feeds
- can create a feed from scratch from local content
- for stored feeds can generate discovery feeds
- a tool for continuous publishing (ingest, seeding) of generated VoD torrent files, automated

updates, additions and removals are supported

- rudimentary ClosedSwarm support: closed swarm keys and corresponding torrent files are generated for manually added items
- Json output: meaningful commands now provide Json output on stdout for integration with other systems (portal, etc.)

Though the second version of the toolbox aims to simplify the implementation, the implementation is still complex. The reason for the complexity are diverse feed sources and a need to consider the requirements of multiple content providers.

The ProviderToolbox was developed on Linux platform (Ubuntu) and it should work on Mac OSX.

At the moment the ProviderToolbox still does not support on the fly generation of P2P-Next Live feeds.

3.2.6.1 Dependencies

The ProviderToolbox depends on JSI's RichMetadata implementation and the existence of ffmpeg . Consult the tool's README for installation. If you have already the RichMetadata implementation installed please update the installation to the latest release version.

Obtain and install ffmpeg tool according to instructions for your target platform. The ffmpeg program should be in your shell path.

Please ensure that all other Next-Share dependencies for your platform are fulfilled, see Next-Share/README.txt for details.

3.2.6.2 Before using the toolbox

Before using the toolbox some adjustments and decisions need to be made to the default settings. The settings are defined in file JSI/ProviderToolbox/conf/default_settings.py.

- decide where fetched and added content to the feeds will be stored using the variable MEDIA_ROOT (default '/media/external'). The directory needs to be writeable and readable by the user running the toolbox.
- decide where the fetched and added content torrent files will be stored using the variable TORRENT_DIR (default /media/external/torrents). If the torrent files will be served via web server the directory should reside in (or it is linked to) the web server's root directory. The directory needs to be writeable and readable by the user running the toolbox.
- decide the internal tracker port and IP using the INTERNAL_TRACKER_PORT and INTERNAL_TRACKER_IP variables. The variables need to be defined before creating any torrents (and fetching or adding any content).

There are a number of other variables set in default_settings.py, the most important are the following:

- the CONTENT_PUBLISHING_LINK variable defines an absolute link for the feed publications (items). The link will be prepend to all your publications in your exported feeds if not defined per feed via the command line.
- the SHADOW_DIR variable controls where the rich metadata in Mpeg7 format of the content is stored. The metadata will appear either as a link or as a content in the feeds

exported. To export it as a link (only automated option) the shadow directory should be accessible through the publishing web server (in web server space, copied, linked, etc.). The shadow inner directory structure is directly appended to CONTENT_PUBLISHING_LINK.

- the EXPORT_FEED_LINK variable defines an absolute link for publishing the feed's xml files. The link will be prepend to all exported feed's xml files, including the discovery feed, if the link is not defined per feed via command line.
- the UPDATE_INTERVAL variable tunes the torrent server update interval. In general, don't set it too low.

To understand how the variables are used please read about the feed life cycle management in the next section.

3.2.6.3 Feed lifecycle management

All the content managed via ProviderToolbox is related to one or more feeds. The feeds and their content could be acquired with the getfeed tool or created with the managefeed tool. Alternatively the feeds can be crafted by hand as well, and placed in the corresponding directories.

The tools briefly described in this section are more broadly discussed in Section 3.2.6.4.

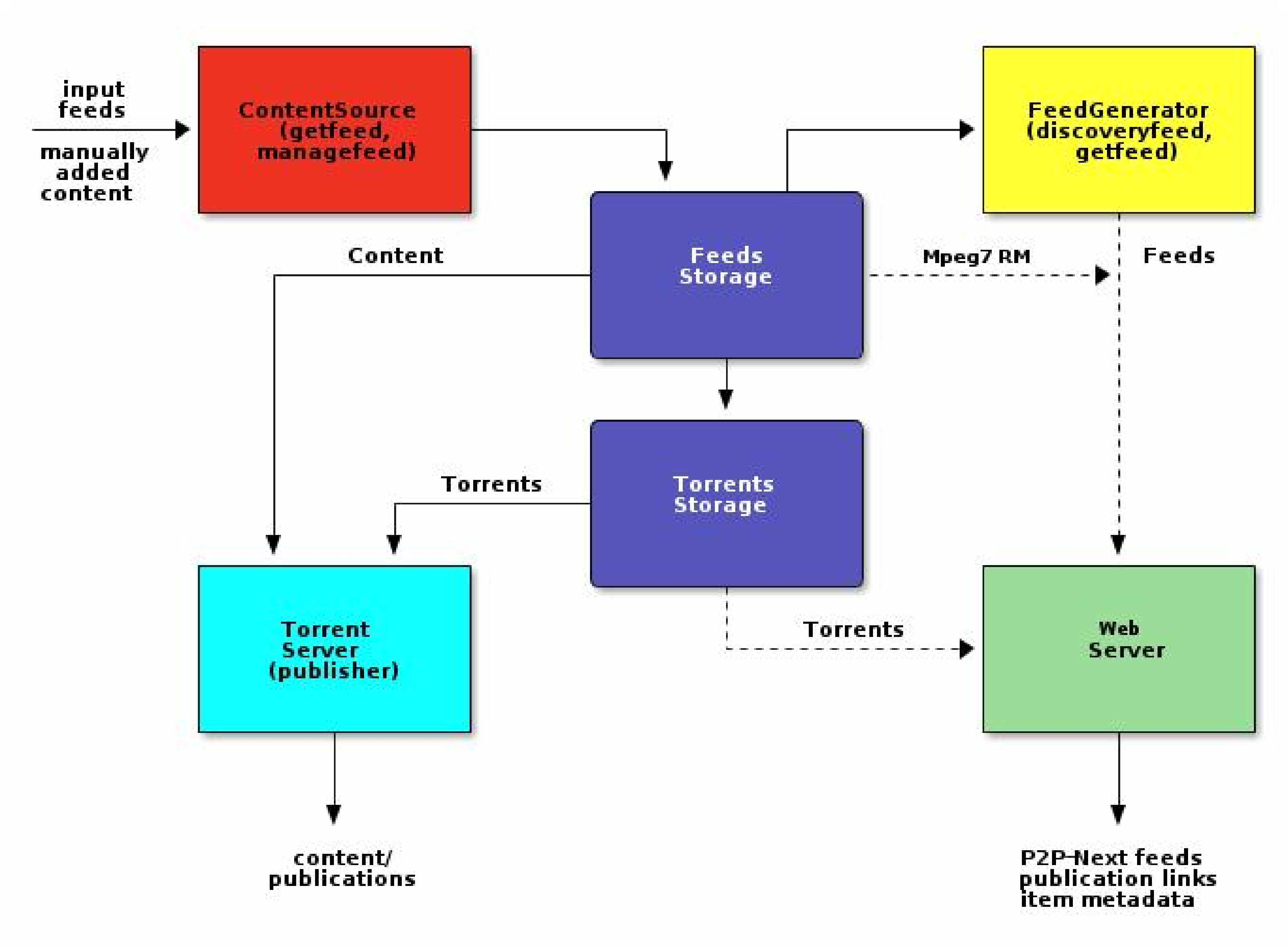

input feeds
manually added content
ContentSource (getfeed, managefeed)
FeedGenerator (discoveryfeed, getfeed)
Content
Feeds Storage
Mpeg7 RM
Feeds
Torrents
Torrents Storage
Torrent Server (publisher)
Torrents
Web Server
content/ publications
P2P-Next feeds publication links item metadata

Figure 11: ProviderToolbox process sketch

If the variables discussed in the previous section are set up properly the system should works as presented in the picture.

The getfeed and managefeed tools (upper left corner) are used to fill the feeds storage. While getting the feed information and feed content the torrent files are generated on the fly containing

P2P-Next compliant rich metadata including with automatically obtained media (content) metadata. The torrent files are stored in the torrents storage. The getfeed tool generates on the standard output the P2P-Next compliant Atom feed for each obtained feed. The generated feeds should be stored in the file system as a file, suitable to be served through web server for user consumption. When the feeds are obtained the discoveryfeed tool can be used to generate a discovery feed from the stored feeds; the tool again outputs the feed on standard output, which should be stored in a file system accessible and served by web server.

The feeds accessed through the web server should be available at an url as defined in getfeed's or discoveryfeed's export option (-e) or as combination of the feed name appended to the variable EXPORT_FEED_LINK as discussed in the previous section.

The publications (content, rich metadata) should be available at an url defined as combination of the absolute publishing link specified via command line and default or via a customized ContentUnit instance specified relative link. Alternatively, a common absolute link can be specified using the CONTENT_PUBLISHING_LINK variable, as discussed in the previous section.

When the feeds are created or acquired they can be published using the publisher tool. The publisher tool acts as a simple seeder and tracker for the content specified in the torrent files. After being started the tool checks the torrent directory (set by the EXPORT_TORRENT_DIR variable) regularly and adds or removes the publications which have been added or removed by getfeed or managefeed.

The only tool not provided at the moment in the toolbox is a web server, at the bottom right in the figure. For serving the feeds and publication links, together with related HTML/Javascript information, a normal web server as provided with your distribution/operating system can be used. On Ubuntu and similar Linux systems, this means that the links to the feeds and publications should point to web server root in '/var/www' or similar. When the feeds are created they should be stored in a suitable place according to theweb server configuration.

3.2.6.3.1 Setting up an example web server

First prepare an example web server for serving the feed xml files and links to the publications. In this example we assume that a Linux system similar to Ubuntu is used and has a similar directory structure as shown:

- /var/www
  - feeds (directory from where the feeds will be served)
  - publish (directory for publications)

Make sure that the user running the ProviderToolbox has write and read permissions to both directories. A series of commands similar to these should prepare the web server, assuming that the web server is already running and operational:

```
sudo mkdir /var/www/feeds
sudo mkdir /var/www/publish
sudo addgroup www
sudo addgroup dusan www
sudo chgrp www /var/www/feeds
sudo chgrp www /var/www/publish
sudo chmod g+rw /var/www/publish
sudo chmod g+rw /var/www/feeds
```

and test the setup, note that you need to log out and log in the system to actually get the www group rights:

```
touch /var/www/publish/lala
```

```
rm /var/www/publish/lala
```

In this section we will refer to this two directories by their names and links as well:

- /var/www/publish: http://stream.e5.ijs.si/publish
- /var/www/feeds: http://stream.e5.ijs.si/feeds

3.2.6.3.2 Adjust default settings

If we would like to serve direct links to torrent files the EXPORT_TORRENT_DIR would need to be changed so it is directly accessible to the user, change the variable in the default_settings.py to:

```
EXPORT_TORRENT_DIR = /var/www/publish
```

After this change the torrents while getting the feed will be written to the specified directory.

3.2.6.3.3 Get the feed initially

Before any publishing the feeds need to be acquired or created. To acquire the feeds use the getfeed tool, for example:

```
xyz:~/src/Next-Share:{1}> python JSI/ProviderToolbox/tools/getfeed.py -l
http://downloads.bbc.co.uk/podcasts/radio4/today/rss.xml
```

The feed can be generated by hand as well, for a lengthy example consult Section 3.2.6.4.3, which shows how the FAB channel content feed could be created.

The result of running the getfeed tool as presented will obtain the content, generate rich metadata for the feed and content and create initial P2P-Next compliant torrent files in the directory as defined in settings.EXPORT_TORRENT_DIR.

Check the results and correct the wrong parameters if any. Consult the tools documentation section if needed.

3.2.6.3.4 Updating the feed

Any feed obtained from the network can be manually updated if needed. In the simplest form, run something like:

```
xyz:~/src/Next-Share:{1}> python JSI/ProviderToolbox/tools/getfeed.py -u
/media/external/Best_of_Today/
```

The directory *media/external/Best_of_Today* is a directory where the initial getfeed run has stored the feed. The feed will be updated, new content acquired, metadata generated and torrents file created.

If the feeds need to be updated automatically some shell script could be helpful, like:

```
#!/bin/sh
NEXT_SHARE_HOME=/home/dusan/delo/src/M32/Next-Share
cd $NEXT_SHARE_HOME
export PYTHONPATH=$(pwd):.
python JSI/ProviderToolbox/tools/getfeed.py -u $1
```

The example script will update the feed specified via the command line parameter, which specifies the directory of the feed. The script can be found in JSI/ProviderToolbox/bin/getfeed. Please note that the variable NEXT_SHARE_HOME needs to be adjusted to your needs.

Run it as:

```
JSI/ProviderToolbox/bin/getfeed /media/external/Best_of_Today
```

and the tool will collect new content, matadata, create the torrent files in the specified directory and output the P2P-Next compliant feed on the standard output.

If you add the path to the bin directory to your PATH in the shell environment the command can be called from wherever you want.

3.2.6.3.4.1 Updating the feed from the cron

After putting the getfeed command in the user PATH the feeds can be added to the users crontab to be run regularly. For example the following line will collect the Best of Today BBC feed every day five minutes after five in the morning:

```
# m h  dom mon dow   command
5 5 * * * getfeed /media/external/Best_of_Today > /var/www/feeds/Best_of_Today.xml
```

Of course as many feeds as needed can be added for collection in the crontab.

3.2.6.3.5 Bring up the torrent server

The torrent server (seeder) can be brought up with the publisher.py tool. Run it from a command line like:

```
xyz:~/src/Next-Share:{1}> python JSI/ProviderToolbox/tools/publisher.py
```

or use the provided shell script in JSI/ProviderToolbox/bin/publisher. If the bin directory is in your path run it like:

```
xyz:~/:{1}> publisher&
```

To run it really independently the scripts need to be detached from the terminal (disown) so that it won't get killed when the terminal is closed.

3.2.6.3.6 Generating a discovery feed

A discovery feed is build based on information of stored feeds. While running the tool you need to specify certain parameters on the command line for proper feed generation, note that the defaults won't work. For example the following command:

```
xyz:~/src/Next-Share:{1}> python JSI/ProviderToolbox/tools/discoveryfeed.py -t "Discovery
feed" -e http://stream.e5.ijs.si/feeds/discovery.xml -p "P2P-Next JSI/RTV Slovenia
(livinglab@e5.ijs.si)" > /var/www/feeds/discovery.xml
```

will generate the discovery feed for collected feeds and redirects the tool standard output to the specified directory accessible by web server.

To automate the process a similar script as was presented in the section on updating the feed should be prepared. In this way updates via cron are possible. It has to be noted that the discoveryfeed tool really needs to be run only when a new feed is added to the feed storage.

3.2.6.3.7 Managing the feed window

The feed window parameter controls how many content items will be kept in the feed while updating.

Each feed window can be managed independently. The default mode of operation is collect, the defined window is None. You can manage the window via specifying the right parameter at the feed creation time with the getfeed tool (-w), on any further run of the same command, or via manually changing the window attribute by editing the feed '.properties' file in the feed storage directory. If the attribute is not specified there add a line like:

```
window = 20
```

to the properties file and the number of the content items published through the feed should be 20.

As said, the default value is None, and the feed is in the collect mode. When the window is 0, the number of items in incoming feed dictates the length of the exported feed. Specifying a negative value for the window on the command line enables reverting to the default collect mode.

3.2.6.3.8 Listing the feeds

The managefeed tool can be used to list the feed content:

```
xyz:~/src/Next-Share:{1}> python JSI/ProviderToolbox/tools/managefeed.py -l
/media/external/Best_of_Today
```

A short listing will provide feed items listed in feed order together with related files and an identifier. The identifier of an item can be used later to remove the item from the feed, if needed.

3.2.6.3.9 Removing the publications or feeds

The publications are removed from the feed by the managefeed tool, specifying the publication (item) identifier:

```
xyz:~/src/Next-Share:{1}> python JSI/ProviderToolbox/tools/managefeed.py -d
/media/external/Best_of_Today -r 625353e5d4000ece34d291c5329ef2a378fdf85c
```

The command removes the content, the content-related rich metadata and the content-related torrent file from the storage.

The feed can be removed by simply removing or moving the feed storage from MEDIA_ROOT e.g, 'rm -rf /media/external/Best_of_Today' or 'mv /media/external/Best_of_Today wherever' will remove the feed. The discovery feed needs to be created again to reflect the change and the feed torrents files need to be removed manually from the torrents storage so the publisher will catch up the change. The files of course won't be shared any more but the publisher will complain in logs about the missing file(s). This information could be helpful if some files torrent files have been forgotten to be deleted.

3.2.6.4 Tools documentation and examples of usage

For documentation consult this description, the README file, the standard python documentation, and the code itself.

3.2.6.4.1 Getfeed tool

The getfeed tool reads single input feed, gets the feed and the embodied media metadata, stores the feed content, generates P2P-Next compliant torrent files and outputs on stdout the P2P-Next compliant VoD feed.

3.2.6.4.1.1 Command line options and their details

The getfeed tool command line help provides the following instructions:

```
xyz:~/src/Next-Share:{1}> python JSI/ProviderToolbox/tools/getfeed.py -h
Usage: getfeed.py [options]

  Gets a feed content and metadata, creates torrent files and outputs P2P-Next
  compliant feed on std out. Consult tool help (-h) for more options.

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -v, --verbose         Be verbose
  -l LOCATION, --location=LOCATION
                        Location of the input feed
  -p PUBLISH, --publish=PUBLISH
```

```
                        Location of the absolute publishing link, default
                        http://stream.e5.ijs.si/publish
  -t TEMPLATE, --content-template=TEMPLATE
                        Name of the content template (class) to use in
                        exporting feed content, for example 'RTVVoDContent'
  -e FEEDEXPORT, --export-link=FEEDEXPORT
                        Location of the exported feed, default
                        http://stream.e5.ijs.si/feeds appended with the name
                        of the feed (directory) with xml extension
  -g GUID, --feed-id=GUID
                        Feed identifier
  -i IMAGE, --feed-image=IMAGE
                        Feed image
  -d DIDBASEFILE, --did-base-file=DIDBASEFILE
                        Common feed DID base file
  -w WINDOW, --window=WINDOW
                        A number of content units to keep, None (default)
                        collect, 0 same as source
  -u DIRECTORY, --update=DIRECTORY
                        Update the feed in specified directory and print fresh
                        feed on std out
  -f FEEDDIR, --feed=FEEDDIR
                        Print the feed of the specified feed directory. Feed
                        guid (-g) or image (-i) can be specified on the
                        command line as well
  -s FEED_EXPORT_STORE, --feed-export-store=FEED_EXPORT_STORE
                        Store the resulting feed in specified directory
                        instead outputing it to stdout. The name of the file
                        will be the same as the name of the feed. Feed export
                        file gets overwritten. Feed guid (-g) or image (-i)
                        can be specified on the command line as well. Useful
                        for command line processing of multiple feeds.
  -r, --fresh           Return instead of the feed the identifiers of the
                        fresh content units are returned, requires update
                        option (-u). The feed can be then obtained with feed
                        option (-f).
  -c, --fresh-content   Return instead of the feed the content files of the
                        fresh content units, requires update option (-u). The
                        feed can be then obtained with feed option (-f).
  -j, --json            Returns a feed data in json format. Used only with
                        location (-l), update (-u) and feed (-f) option.
```

3.2.6.4.1.1.1 Location option

The '-l' option specifies the location of input feed that will be read and from which metadata and content will be obtained.

Both matadata and content will be stored under a directory of your choice, specified using the settings.MEDIA_ROOT variable. See conf/default_settings.py for more details.

Example layout of such storage, MEDIA_ROOT points to '/media/external':

```
  xyz:~/src/Next-Share:{1}> ls /media/external/*

/media/external/Best_of_Today:
Best_of_Today.xml         today_20101111-0652a.mp3  today_20101112-0923a.xml
today_20101109-0641a.mp3  today_20101111-0652a.xml  today_20101113-0840a.mp3
today_20101109-0641a.xml  today_20101111-0915a.mp3  today_20101113-0840a.xml
today_20101109-0921a.mp3  today_20101111-0915a.xml  today_20101113-1057a.mp3
today_20101109-0921a.xml  today_20101111-0918a.mp3  today_20101113-1057a.xml
today_20101109-0926a.mp3  today_20101111-0918a.xml  today_20101115-0757a.mp3
today_20101109-0926a.xml  today_20101111-0923a.mp3  today_20101115-0757a.xml
today_20101110-0643a.mp3  today_20101111-0923a.xml  today_20101115-0804a.mp3
today_20101110-0643a.xml  today_20101112-0645a.mp3  today_20101115-0804a.xml
today_20101110-0946a.mp3  today_20101112-0645a.xml  today_20101115-0954a.mp3
today_20101110-0946a.xml  today_20101112-0917a.mp3  today_20101115-0954a.xml
today_20101110-0955a.mp3  today_20101112-0917a.xml
today_20101110-0955a.xml  today_20101112-0923a.mp3

/media/external/RTV_-_Zapisi_iz_Mocvirja:
RTV_-_Zapisi_iz_Mocvirja.xml            zapisi_iz_mocvirja_09-11-2010_1629.xml
zapisi_iz_mocvirja_02-11-2010_1629.mp3  zapisi_iz_mocvirja_26-10-2010_1628.mp3
```

```
zapisi_iz_mocvirja_02-11-2010_1629.xml  zapisi_iz_mocvirja_26-10-2010_1628.xml
zapisi_iz_mocvirja_09-11-2010_1629.mp3
```

The storage contains two feeds stored in directories derived from the feed names. Each unit of content is present with two files, a content file and metadata file. It is needles to say that the user invoking the getfeed script should have a write permission to the MEDIA_ROOT or feed directory. The feed directories are created on fly, if not existent, and read, and content and metadata information restored, if present. Restored information is used in updates of the feed to obtain only fresh content (but the whole feed is read from the network).

3.2.6.4.1.1.2 Publish option

The publish option '-p' specifies an absolute path of the location where the publications could be read as obtained from the exported P2P-Next feed. If you plan to publish, for example, torrent files of the input feed content through a web server like http://web.server.of.your.choice/publications, this is the absolute path that will appear in the feed as a link to the publication. The path could be not specified as well. To get the final link to the publication while generating the feed the absolute path is concatenated with the content's (feed item) relative path as explained in the next section.

For simpler managing of this option you can specify a default value in default_settings.py using the variable CONTENT_PUBLISHINGLINK. Note that if you are not specifying this variable for every feed, it will be used for all your feeds.

3.2.6.4.1.1.3 Export url option

The export url option (-e) defines where the resulting feed will be obtainable from. Usually it will be obtained from a web server, therefore the link should point to there.

For simpler managing of the feeds you can specify a default value in default_settings.py using the variable EXPORT_FEEDLINK. In this case the the variable will be appended with the feed name and the xml extension by default. Be sure that the file is stored with the right name in your web server. Note that if you are not specifying a location for every feed, this variable will be used for all your feeds.

- Template option

By default the absolute path is concatenated with the content's relative path while building the final path for the content. The relative path is a name of the content's torrent file. The default behavior can be changed by providing your customized ContentUnit inherited class in the ContentSource.py module. There are three methods that were intended for customization in the ContentUnit class, presented here together with their help:

```
def getImage(self):
    """
    Per content unit settable image. Define any mapping on content
    unit attributes ta return the image location as a string. If
    the method returns None content unit image will equal to feed
    image.

    @return string String or None if not defined.
    """
    return None

def getId(self):
    """
    Per content unit settable id. Define any mapping on content
    unit attributes ta return the content id as a string. If the
    method returns None content unit id will be equal to its link.
```

```
        @return string String or None if not defined.
        """
        return None

    def getPublish(self):
        """
        Per content unit settable relative publishing link. Define any
        mapping on content unit attributes ta return the content
        relative publishing link as a string. If the method returns
        None content unit relative publishing link will be equal to
        its torrent publication.

        @return string String or None if not defined.
        """
        return None
```

They allow programmable manipulation of the content related parameters. See RTVVoDContent class in ContentSource.py for a very simple implementation of such a template.

3.2.6.4.1.1.4 Feed uid and image options

The getfeed tool options '-g' and '-i' allow specifying the feed's unique id and image. If '-g' is not defined as unique feed id, the feed's exported link is used (see next section). If '-i' is not specified, the original output feed image is used.

- DIDBase option

The DIDbase option '-d' enables providing a path to the MPEG-21 DID base file as input parameter. The DIDbase provided in this way defines the same DIDbase information for all content units in the feed. To specify the DIDbase for each content unit separately one should overload the setDIDBase method in a class inheriting from ContentUnit class (similarly as explained in the section on Template option, example not provided). The DIDbase file can be generated with a help of the metagen tool provided by the RichMetadata implementation. Not all options need to be specified, for example the following command will provide enough information for generating a complete DIDbase for a torrent file:

python JSI/RichMetadata/tools/metagen.py -b –identifier="urn:p2p-next:item:rtv-slo-slo1-xyz" –relatedIdentifier="urn:rtv-slo:slo1-xyz" –advertisementReference="URI to additional MPEG$_{21}$ data (advertising)" –limoReference="URI to additional MPEG$_{21}$ data (limo)" –paymentReference="URI to additional MPEG$_{21}$ data (payment)" –scalabilityReference="URI to additional MPEG$_{21}$ data (scalability)" -c > did-base.xml

Other missing parameters like metadata core, etc,. are already available after getting the original feed metadata information.

Without providing the DIDbase option, the only available elements based on original feed are packed in the DID base and included in the torrent file.

The torrent files as created are stored as specified by settings.EXPORT_TORRENT_DIR in default_settings.py. Per default this directory is '/media/external/torrents', for example torrent files for the feeds as were presented above are collected in a single directory:

```
/media/external/torrents:
today_20101109-0641a.tstream  today_20101112-0917a.tstream
today_20101109-0921a.tstream  today_20101112-0923a.tstream
today_20101109-0926a.tstream  today_20101113-0840a.tstream
today_20101110-0643a.tstream  today_20101113-1057a.tstream
today_20101110-0946a.tstream  today_20101115-0757a.tstream
today_20101110-0955a.tstream  today_20101115-0804a.tstream
today_20101111-0652a.tstream  today_20101115-0954a.tstream
today_20101111-0915a.tstream  zapisi_iz_mocvirja_02-11-2010_1629.tstream
today_20101111-0918a.tstream  zapisi_iz_mocvirja_09-11-2010_1629.tstream
```

```
today_20101111-0923a.tstream  zapisi_iz_mocvirja_26-10-2010_1628.tstream
today_20101112-0645a.tstream
```

It has to be noted that currently no resolving mechanism exists if there are two content units with the same name published through one or more feeds - the resulting torrent file will be overwritten by the torrent file of the content with the same name acquired latter.

3.2.6.4.1.1.5 Update option

For easier updating of the feeds when already created you can specify just the update option (-u), followed by the storage directory of the feed. The feed will be updated, using the same parameters or defaults, as when they were created. The resulting feed will be directed to the standard output. If the fresh option (-r) or fresh content option (-c) are specified the output will consist of fresh content units item identifiers or content files. Use the feed option (-f) to get the atom feed or specify the update with the Json option (-j) and get all the specified data on standard output. See the Json option section for details.

3.2.6.4.1.1.6 Feed option

The feed option outputs the feed as atom feed on standard output.

3.2.6.4.1.1.7 Json outputs

Json output can ease using the getfeed tool wrapped as a shell script. The Json output provides the following data, when used with update (-u), location (-l) or feed option (-f):

```
{"feed": atom feed,
"fresh": fresh items,
"maps": { metadata_file_name: { "content": content,
                                 "torrent": torrent file,
                                 "id": id of the item in atom feed,
                                 "identifier": identifier of the item,
                                 "cskeys": ClosedSwarm keys}
```

The maps value is a dictionary of metadata file names as keys and dictionaries about the item details, as specified above as a value. Fresh items will be available only when Json output is used with the update or location option. ClosedSwarm keys will be listed in the output if the content distribution is protected with ClosedSwarm technology. Similar Json outputs could be provided by the managefeed tool as well.

3.2.6.4.1.2 Quick summary

To obtain a feed from a location and publish it on a web server of your choice:

```
xyz:~/src/Next-Share:{1}> python JSI/ProviderToolbox/tools/getfeed.py -l
http://www.rtvslo.si/podcasts/zapisi_iz_mocvirja.xml -p
http://web.server.of.your.choice/publications
```

To modify as well the location of the exported feed, e.g. where the exported feed can be obtained from (link that appears in the feed itself):

```
xyz:~/src/Next-Share:{1}> python JSI/ProviderToolbox/tools/getfeed.py -l
http://www.rtvslo.si/podcasts/zapisi_iz_mocvirja.xml -p
http://web.server.of.your.choice/publications -e
http://web.server.of.your.choice/feeds/myfeed.xml
```

To use a customize template is enough to specify the right ContentUnit class:

```
xyz:~/src/Next-Share:{1}> python JSI/ProviderToolbox/tools/getfeed.py -l
http://www.rtvslo.si/podcasts/zapisi_iz_mocvirja.xml -p
http://web.server.of.your.choice/publications -e
http://web.server.of.your.choice/feeds/myfeed.xml -t RTVVoDContent
```

To update the feed:

```
xyz:~/src/Next-Share:{1}> python JSI/ProviderToolbox/tools/getfeed.py -l
http://www.rtvslo.si/podcasts/zapisi_iz_mocvirja.xml
```

or even simpler:

```
xyz:~/src/Next-Share:{1}> python JSI/ProviderToolbox/tools/getfeed.py -u
/media/external/RTV_-_Zapisi_iz_Mocvirja
```

3.2.6.4.2 Discoveryfeed tool

The discoveryfeed tool creates a P2P-Next compliant discovery feed from the stored feeds and outputs it on stdout.

3.2.6.4.2.1 Command line options and their details

The discoveryfeed tool command line help provides the following instructions:

```
  xyz:~/src/Next-Share:{1}> python JSI/ProviderToolbox/tools/getfeed.py


Usage: discoveryfeed.py [options]

  Create discoveryfeed from scratch. Consult tool help (-h) for more options.

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -v, --verbose         Be verbose
  -m MEDIA_ROOT, --media-root=MEDIA_ROOT
                        Location of feeds storage, other then media root
                        (settings.MEDIA_ROOT)
  -t TITLE, --feed-title=TITLE
                        Feed title
  -e EXPORTURL, --feed-export-url=EXPORTURL
                        URL where the feed will be accessible
  -p PUBLISHER, --author=PUBLISHER
                        Publisher of the feed
  -u ID, --id=ID        Feed unique identifier
  -i IMAGE, --image=IMAGE
                        Feed image
```

Please note that the output depends on the stored feeds. the Discovery feed items' information is based on the stored feed information. If there are no feeds stored the tool will return only the feed header.

3.2.6.4.2.2 Quick summary

An example of usage:

```
xyz:~/src/Next-Share:{1}> python JSI/ProviderToolbox/tools/discoveryfeed.py -t "Discovery
feed" -e http://stream.e5.ijs.si/discovery.xml -p "P2P-Next JSI/RTV Slovenia
(livinglab@e5.ijs.si)"
```

3.2.6.4.3 Managefeed tool

The managefeed tool enables creating a feed from scratch. While the getfeed tool creates the feed from a source in a network, managefeed needs proper inputs for feed creation. Both type of feeds are stored in a same manner in the file system with the same or similar properties.

The managefeed tool main operations supported are creation of a feed, adding to and removing items from the feed and list the feed. Since the information about the feed and its items is not obtained from an external source the number of parameters supported (and needed for meaningful management) by the tool is significantly increased. The command line help for the tool returns:

```
  xyz:~/src/Next-Share:{1}> python JSI/ProviderToolbox/tools/managefeed.py -h

Reason for failure: None of the main options -c, -a, -r, -l, -f or --identifier specified!
```

```
Usage: managefeed.py [options]

  Creates a feed from scratch or manages the feed. Consult tool help (-h)
  for more options.

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -v, --verbose         Be verbose

  Main options:
    One of this options needs to be specified

    -c, --create-feed   Create or modify feed metadata or parameters, the feed
                        directory should be specified as option argument
    -a, --add-item      Add feed item. The feed should be specified with feed
                        dir option (-d)
    -r REMOVEITEM, --remove-item=REMOVEITEM
                        Remove feed item by identifier (use list to find the
                        right one and specify the feed directory with -d)
    -l LIST, --list=LIST
                        List the feed storage directory as specified in option
    -f FEED, --feed=FEED
                        Get the feed specified by the feed storage directory.
                        Feed guid (-u) or image (-i) can be specified on the
                        command line as well
    -d FEEDDIR, --feed-storage=FEEDDIR
                        Feed storage. Not usable on its own but needs to be
                        specified together with other options (-r, -a,
                        --identifier, ---find-cs-key)

  Metadata options:
    Options used for specifying the feed or item metadata

    -t TITLE, --title-name=TITLE
                        Name of the title, feed or item
    -k DESCRIPTION, --series-title=DESCRIPTION
                        Feed series title (description)
    -n LANGUAGE, --language=LANGUAGE
                        Feed language
    -g ORIGINATOR, --originator=ORIGINATOR
                        Feed originator
    -j PUBLISHER, --author=PUBLISHER
                        Publisher of the feed
    -x COREMETAFILE, --core-metadata=COREMETAFILE
                        Core metadata file used as template for the feed or an
                        item. Caution: if used with the item the channel
                        metadata gets overwritten by this metadata. In this
                        case the item metadata should specify the channel
                        metadata as well.
    -s SYNOPSIS, --synopsis=SYNOPSIS
                        Item synopsis

  Publishing options:
    Options used for publishing the content

    -e EXPORTURL, --feed-export-url=EXPORTURL
                        URL where the feed will be accessible, default
                        http://stream.e5.ijs.si/feeds appended with the name
                        of the feed (directory) with xml extension
    -u ID, --id=ID      Unique identifier of the feed
    -i IMAGE, --image=IMAGE
                        Image of the feed
    -p PUBLISH, --publish-link=PUBLISH
                        Publish link of the feed (absolute), default
                        http://stream.e5.ijs.si/publish
    -o CUCI, --cu-class-instance=CUCI
                        Content unit class instance used in an export of the
                        feed. Enables custumization of the export per unit
    -b DIDBASE, --did-base=DIDBASE
                        Feed DID base file
    -y MIMETYPE, --mime-type=MIMETYPE
                        Item mime type
    -z CONTENT, --content=CONTENT
                        Content file pointed to in an item

  ClosedSwarm options:
    --cs                Protect the item with the closed swarm. Used only with
                        add item (-a)
    --cs-keys=CSKEYS    Specify a comma separated list of keys to be used with
                        closed swarm. Used only with add item (-a). Not tested
                        yet.
    --find-cs-key=IDENTIFIER
                        Find CS key file for specified content unit
                        identifyer, if any. Use with feed directory (-d)

  Miscellaneous options:
    --long              List the feed storage in detail
    -m MEDIA_ROOT, --media-root=MEDIA_ROOT
                        Location of feeds storage, other then media root
                        (settings.MEDIA_ROOT)
    --identifier=FILENAME
                        Return an identifier list (comma separated) according
                        to specified file name, either content, metadata or
                        torrent file. Requires an option feed storage (-d) as
                        well.
    --json              Output feed data in json format. Makes sense only in
                        combination with list feed (-l), feed (-f),
                        --identifier and --find-cs-key.
```

The first line warns us why the failure has occurred and then the tool help is returned. One of the main modes of the tool needs to be specified:

- '-c': create the feed from scratch
- '-a': add a new item to the feed
- '-r': remove the item from the feed, or
- '-l': list the feed information
- '-f': gets the feed atom feed without an update
- '–identifier': utility option

The other parameters' usage will be explained through examples.

3.2.6.4.3.1 Create a feed

The feed can be created as presented in the following example. The content feed for FAB channel is created with a suitable title and description. RichMetadata-wise they get mapped to the main and the series title. A feed image is assigned, an url where the feed could be obtained from (-e), and an absolute publishing link for the feed items (-p):

```
xyz:~/src/Next-Share:{1}> python JSI/ProviderToolbox/tools/managefeed.py -c -t "FAB Channel
feed" -k "Concerts in Paradiso, Amsterdam" -g "FAB channel" -j "P2P-Next (www.p2p-next.org)"
-n "en" -i "http://stream.e5.ijs.si/images/p2p-next-logo.jpg" -e
"http://stream.e5.ijs.si/feeds/fab.xml" -p "http://stream.e5.ijs.si/torrents"
```

The result of the command is a new feed storage directory at settings.MEDIA_ROOT, in this case '*media/external*':

```
  xyz:~/src/Next-Share:{1}> ls -a /media/external/FAB_Channel_feed
.  ..  FAB_Channel_feed.xml  .properties
```

The xml file holds the feed's RichMetadata information and the .properties file's feed properties:

```
  xyz:~/src/Next-Share:{1}> python JSI/ProviderToolbox/tools/printxml.py -f
/media/external/FAB_Channel_feed/FAB_Channel_feed.xml
<?xml version="1.0" encoding="utf-8"?>
<TVAMain      publisher="P2P-Next      (www.p2p-next.org)"      xmlns="urn:tva:metadata:2007"
xmlns:mpeg7="urn:mpeg:mpeg7:schema:2001"                 xmlns:mpeg7_tva="urn:tva:mpeg7:2005"
xmlns:p2pnext="urn:p2pnext:metadata:2008"                   xmlns:tva="urn:tva:metadata:2007"
```

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:tva:metadata:2007 tva_metadata_3-1_v141_p2p.xsd">
   <ProgramDescription>
      <ProgramInformationTable>
         <ProgramInformation>
            <BasicDescription type="p2pnext:BasicP2PDataDescriptionType">
               <Title type="main">
                  FAB Channel feed
               </Title>
               <Title type="seriesTitle">
                  Concerts in Paradiso, Amsterdam
               </Title>
               <Language>
                  en
               </Language>
               <p2pnext:Originator>
                  FAB channel
               </p2pnext:Originator>
            </BasicDescription>
         </ProgramInformation>
      </ProgramInformationTable>
   </ProgramDescription>
</TVAMain>
  xyz:~/src/Next-Share:{1}> cat /media/external/FAB_Channel_feed/.properties
location = file:///media/external/FAB_Channel_feed
name = FAB Channel feed
cstype = channel
publish = http://stream.e5.ijs.si/torrents
image = http://stream.e5.ijs.si/images/p2p-next-logo.jpg
contentUnitClassInstance = ContentUnit
exportFeedLink = http://stream.e5.ijs.si/feeds/fab.xml
```

If the same feed with the same name is created with another '-c' command, the feed information is simply overwritten.

3.2.6.4.3.2 Add item to the feed

Subsequently, the items of the feed can be added to the feed. Lets say that we would like to add the FAB channel content as available in the directory '/media/external/content/20050526_monokino.m4v'. The following command will add an item to the feed and create the item's RichMetadata from the feed metadata, the information supplied on the command line and the technical metadata of the content:

```
xyz:~/src/Next-Share:{1}> python JSI/ProviderToolbox/tools/managefeed.py -a -d
/media/external/FAB_Channel_feed -z /media/external/content/20081213_gpnl_missippi_hdv.m4v -s
"Missippi whatever synopsis" -t "Misippi live performance in Paradiso, Amsterdam, 2008"
```

The result will show up in the feed storage:

```
  xyz:~/src/Next-Share:{1}> ls -al /media/external/FAB_Channel_feed
lrwxrwxrwx 1 dusan dusan   54 2010-11-22 11:43 20081213_gpnl_missippi_hdv.m4v ->
/media/external/content/20081213_gpnl_missippi_hdv.m4v
-rw-r--r-- 1 dusan dusan 1476 2010-11-22 11:43 20081213_gpnl_missippi_hdv.xml
-rw-r--r-- 1 dusan dusan  766 2010-11-22 11:30 FAB_Channel_feed.xml
-rw-r--r-- 1 dusan dusan  286 2010-11-22 11:30 .properties
```

As can be seen the content is simply a link to the content specified on the command line (the item can be easily removed without loosing the content itself) and the content metadata:

```
  xyz:~/src/Next-Share:{1}> python JSI/ProviderToolbox/tools/printxml.py -f
/media/external/FAB_Channel_feed/20081213_gpnl_missippi_hdv.xml

<?xml version="1.0" encoding="utf-8"?>
<TVAMain publisher="P2P-Next (www.p2p-next.org)" xmlns="urn:tva:metadata:2007"
xmlns:mpeg7="urn:mpeg:mpeg7:schema:2001" xmlns:mpeg7_tva="urn:tva:mpeg7:2005"
xmlns:p2pnext="urn:p2pnext:metadata:2008" xmlns:tva="urn:tva:metadata:2007"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:tva:metadata:2007 tva_metadata_3-1_v141_p2p.xsd">
   <ProgramDescription>
      <ProgramInformationTable>
         <ProgramInformation>
            <BasicDescription type="p2pnext:BasicP2PDataDescriptionType">
```

```
               <Title type="main">
                  FAB Channel feed
               </Title>
               <Title type="seriesTitle">
                  Concerts in Paradiso, Amsterdam
               </Title>
               <Title type="episodeTitle">
                  Misippi live performance in Paradiso, Amsterdam, 2008
               </Title>
               <Synopsis>
                  Missippi whatever synopsis
               </Synopsis>
               <Language>
                  en
               </Language>
               <Duration>
                  00:16:40.17
               </Duration>
               <p2pnext:Originator>
                  FAB channel
               </p2pnext:Originator>
            </BasicDescription>
            <AVAttributes>
               <FileFormat href="urn:mpeg:mpeg7:cs:FileFormatCS:2001">
                  <Name>
                     mov
                  </Name>
               </FileFormat>
               <FileSize>
                  1564901380
               </FileSize>
               <BitRate>
                  12517 kb/s
               </BitRate>
               <AudioAttributes>
                  <Coding href="urn:mpeg:mpeg7:cs:AudioCodingFormatCS:2001">
                     <Name>
                        aac
                     </Name>
                  </Coding>
                  <NumOfChannels>
                     2
                  </NumOfChannels>
               </AudioAttributes>
               <VideoAttributes>
                  <Coding href="urn:mpeg:mpeg7:cs:VisualCodingFormatCS:2001">
                     <Name>
                        h264
                     </Name>
                  </Coding>
                  <HorizontalSize>
                     1280
                  </HorizontalSize>
                  <VerticalSize>
                     720
                  </VerticalSize>
                  <AspectRatio>
                     16:9
                  </AspectRatio>
               </VideoAttributes>
            </AVAttributes>
         </ProgramInformation>
      </ProgramInformationTable>
   </ProgramDescription>
</TVAMain>
```

As can be seen from the provided example the feed metadata gets mapped to main and series title and the item title to the episode title. The technical metadata gets extracted from the content automatically.

At the same time the torrent file was created for the item. It is located at the location specified with the settings.EXPORT_TORRENT_DIR variable (default '/media/external/torrents'). We can take a look at the torrent file created with the RichMetadata tool btshowmetainfo:

```
   xyz:~/src/Next-Share:{1}> python JSI/RichMetadata/tools/btshowmetainfo.py
/media/external/torrents/20081213_gpnl_missippi_hdv.tstream

metainfo: ['creation date', 'announce', 'info', 'azureus_properties', 'encoding']
azprop: ['Content']
content: ['Publisher', 'Description', 'Progressive', 'Title', 'Creation Date', 'Content
Hash', 'Speed Bps', 'Revision Date']
Publisher = Tribler
Description =
Progressive = 1
Title = 20081213_gpnl_missippi_hdv.m4v
Creation Date = 1290422609
Content Hash = PT3GQCPW4NPT6WRKKT25IQD4MU5HM4UY
Speed Bps = 1564901
Revision Date = 1290422609
metainfo file.: 20081213_gpnl_missippi_hdv.tstream
info hash.....: d1ed1996375d63de38074e9e1f835973105323c2
info hash.....: '\xd1\xed\x19\x967]c\xde8\x07N\x9e\x1f\x83Ys\x10S#\xc2'
file name.....: 20081213_gpnl_missippi_hdv.m4v
file size.....: 1564901380 (47757 * 32768 + 4)
announce url..: http://127.0.0.1:6969/announce
ns-metadata...:
<?xml version="1.0" ?>
<DIDL xmlns:didl="urn:mpeg:mpeg21:2002:02-DIDL-NS" xmlns:dii="urn:mpeg:mpeg21:2002:01-DII-NS"
xmlns:xi="http://www.w3.org/2001/XInclude" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="urn:mpeg:mpeg21:2002:02-DIDL-NS didl.xsd
urn:mpeg:mpeg21:2002:01-DII-NS dii.xsd">
   <Item>
      <Descriptor>
         <Statement mimeType="text/xml">
            <dii:Type>
               urn:p2p-next:type:item:2009
            </dii:Type>
         </Statement>
      </Descriptor>
      <Descriptor>
         <Descriptor>
            <Statement mimeType="text/xml">
               <dii:Type>
                  urn:p2p-next:type:rm:core:2009
               </dii:Type>
            </Statement>
         </Descriptor>
         <Statement mimeType="text/xml">
            &lt;TVAMain publisher=&quot;P2P-Next (www.p2p-next.org)&quot;
xmlns=&quot;urn:tva:metadata:2007&quot; xmlns:mpeg7=&quot;urn:mpeg:mpeg7:schema:2001&quot;
xmlns:mpeg7_tva=&quot;urn:tva:mpeg7:2005&quot;
xmlns:p2pnext=&quot;urn:p2pnext:metadata:2008&quot;
xmlns:tva=&quot;urn:tva:metadata:2007&quot; xmlns:xsi=&quot;http://www.w3.org/2001/XMLSchema-
instance&quot; xsi:schemaLocation=&quot;urn:tva:metadata:2007 tva_metadata_3-
1_v141_p2p.xsd&quot;&gt;&lt;ProgramDescription&gt;&lt;ProgramInformationTable&gt;&lt;ProgramI
nformation&gt;&lt;BasicDescription
type=&quot;p2pnext:BasicP2PDataDescriptionType&quot;&gt;&lt;Title
type=&quot;main&quot;&gt;FAB Channel feed&lt;/Title&gt;&lt;Title
type=&quot;seriesTitle&quot;&gt;Concerts in Paradiso, Amsterdam&lt;/Title&gt;&lt;Title
type=&quot;episodeTitle&quot;&gt;Misippi live performance in Paradiso, Amsterdam,
2008&lt;/Title&gt;&lt;Synopsis&gt;Missippi whatever
synopsis&lt;/Synopsis&gt;&lt;Language&gt;en&lt;/Language&gt;&lt;Duration&gt;00:16:40.17&lt;/D
uration&gt;&lt;p2pnext:Originator&gt;FAB
channel&lt;/p2pnext:Originator&gt;&lt;/BasicDescription&gt;&lt;AVAttributes&gt;&lt;FileFormat
href=&quot;urn:mpeg:mpeg7:cs:FileFormatCS:2001&quot;&gt;&lt;Name&gt;mov&lt;/Name&gt;&lt;/File
Format&gt;&lt;FileSize&gt;1564901380&lt;/FileSize&gt;&lt;BitRate&gt;12517
kb/s&lt;/BitRate&gt;&lt;AudioAttributes&gt;&lt;Coding
href=&quot;urn:mpeg:mpeg7:cs:AudioCodingFormatCS:2001&quot;&gt;&lt;Name&gt;aac&lt;/Name&gt;&l
t;/Coding&gt;&lt;NumOfChannels&gt;2&lt;/NumOfChannels&gt;&lt;/AudioAttributes&gt;&lt;VideoAtt
ributes&gt;&lt;Coding
href=&quot;urn:mpeg:mpeg7:cs:VisualCodingFormatCS:2001&quot;&gt;&lt;Name&gt;h264&lt;/Name&gt;
&lt;/Coding&gt;&lt;HorizontalSize&gt;1280&lt;/HorizontalSize&gt;&lt;VerticalSize&gt;720&lt;/V
erticalSize&gt;&lt;AspectRatio&gt;16:9&lt;/AspectRatio&gt;&lt;/VideoAttributes&gt;&lt;/AVAttr
ibutes&gt;&lt;/ProgramInformation&gt;&lt;/ProgramInformationTable&gt;&lt;/ProgramDescription&
gt;&lt;/TVAMain&gt;
         </Statement>
      </Descriptor>
      <Component>
```

```
            <Resource mimeType="video/x-m4v" ref="20081213_gpnl_missippi_hdv.m4v"/>
         </Component>
      </Item>
   </DIDL>
```

Another item could be added with a similar command:

```
xyz:~/src/Next-Share:{1}> python JSI/ProviderToolbox/tools/managefeed.py -a -d
/media/external/FAB_Channel_feed -z /media/external/content/20050526_monokino.m4v -s
"Monokino whatever synopsis" -t "Monokino live performance in Paradiso, Amsterdam, 2005"
```

The managefeed tool command line parameters allow specifying only parameters essential for the feed creation. If there is a need for additional information in the RichMetadata of the content, like genre, minimum age, etc., one can create the necessary metadata before creating a feed and adding an item to the feed. For example, the feed metadata could be defined with the RichMetadata metagen tool:

```
xyz:~/src/Next-Share:{1}> python JSI/RichMetadata/tools/metagen.py --captionLanguage=en
--genre="Live concert" --language=en --minimumAge=3 --originator="FAB Channel"
--productionDate=2010-08-16 --productionLocation=NL --publisher=p2p-next --releaseDate=2010-
08-17 --signLanguage=en --titleMain="FAB Channel feed" --titleSeriesTitle="Concerts in
Paradiso, Amsterdam" > fab.xml

xyz:~/src/Next-Share:{1}> python JSI/ProviderToolbox/tools/managefeed.py -c -i
"http://stream.e5.ijs.si/images/p2p-next-logo.jpg" -e "http://stream.e5.ijs.si/feeds/fab.xml"
-p "http://stream.e5.ijs.si/torrents" -x fab.xml
```

and provided as an template while creating the feed ('-x' option). The metadata can be replaced directly by copying the metadata to the feed store (in this case replacing the file FAB_Channel_Feed.xml). All items added to the feed later will inherit the same data as stored in this file, except if you use the template while adding the item as well. In this case the item metadata should include the feed metadata as well:

```
xyz:~/src/Next-Share:{1}> python JSI/RichMetadata/tools/metagen.py --captionLanguage=en
--genre="Live concert" --language=en --minimumAge=3 --originator="FAB Channel"
--productionDate=2010-08-16 --productionLocation=NL --publisher=p2p-next --releaseDate=2010-
08-17 --signLanguage=en --synopsis="Monokino whatever synopsis" --titleEpisodeTitle="Monokino
live performance in Paradiso, Amsterdam, 2005" --titleMain="FAB Channel feed"
--titleSeriesTitle="Concerts in Paradiso, Amsterdam" > fab-monokino.xml
```

and the template can be used while creating an item:

```
xyz:~/src/Next-Share:{1}> python JSI/ProviderToolbox/tools/managefeed.py -a -d
/media/external/FAB_Channel_feed -z /media/external/content/20081213_gpnl_missippi_hdv.m4v -x
fab-monokino.xml
```

If the resulted feed is now listed, we see two items in the feed:

```
  xyz:~/src/Next-Share:{1}> python JSI/ProviderToolbox/tools/managefeed.py -l
/media/external/FAB_Channel_feed

'FAB Channel feed', stored in /media/external/FAB_Channel_feed
Location: file:///media/external/FAB_Channel_feed
  1) Monokino live performance in Paradiso, Amsterdam, 2005
     Identifier: 42d63e76f63c979a5d63458a2867606358e229a1
     Content:    20050526_monokino.m4v
     Metadata:   20050526_monokino.xml
     Torrent:    20050526_monokino.tstream
  2) Misippi live performance in Paradiso, Amsterdam, 2008
     Identifier: 8a9640c940c4958d863797802d50d3ff5029b8bb
     Content:    20081213_gpnl_missippi_hdv.m4v
     Metadata:   20081213_gpnl_missippi_hdv.xml
     Torrent:    20081213_gpnl_missippi_hdv.tstream
```

We can see in the list output the item identifiers as well. They can be used for removing the items from the feed:

```
xyz:~/src/Next-Share:{1}> python JSI/ProviderToolbox/tools/managefeed.py -d
/media/external/FAB_Channel_feed -r 8a9640c940c4958d863797802d50d3ff5029b8bb
```

would remove the Missippi item from the feed.

3.2.6.4.3.3 Getting the atom feed

The P2P-Next compliant feed could then be obtained from the created feed with the getfeed or the managefeed tool:

```
  xyz:~/src/Next-Share:{1}> python JSI/ProviderToolbox/tools/getfeed.py -l
/media/external/FAB_Channel_feed
<?xml version="1.0" encoding="utf-8"?>
<feed xml:lang="en" xmlns="http://www.w3.org/2005/Atom"
xmlns:p2pnext="urn:p2pnext:contentfeed:2009">
   <title>
      FAB Channel feed
   </title>
   <link href="http://stream.e5.ijs.si/feeds/FAB_Channel_feed.xml" rel="self"/>
   <id>
      http://stream.e5.ijs.si/feeds/FAB_Channel_feed.xml
   </id>
   <updated>
      2011-04-04T16:37:40Z
   </updated>
   <author>
      <name>
         P2P-Next (www.p2p-next.org)
      </name>
   </author>
   <p2pnext:image src="http://stream.e5.ijs.si/images/p2p-next-logo.jpg"/>
   <entry>
      <title>
         Misippi live performance in Paradiso, Amsterdam, 2008
      </title>
      <link
href="http://stream.e5.ijs.si/publish/FAB_Channel_feed/20081213_gpnl_missippi_hdv"
rel="alternate" type="application/xml"/>
      <updated>
         2011-04-04T16:37:40Z
      </updated>
      <id>
         urn:p2p-
next:item:tag:stream.e5.ijs.si:/publish/FAB_Channel_feed/20081213_gpnl_missippi_hdv/
      </id>
      <summary>
         Mississipi concert
      </summary>
      <p2pnext:image src="http://stream.e5.ijs.si/images/p2p-next-logo.jpg"/>
      <p2pnext:broadcastType>
         vod
      </p2pnext:broadcastType>
      <p2pnext:mediaUri>
         http://stream.e5.ijs.si/publish/torrents/20081213_gpnl_missippi_hdv.tstream
      </p2pnext:mediaUri>
      <p2pnext:mediaDuration>
         PT00H16M40S
      </p2pnext:mediaDuration>
   </entry>
   <entry>
      <title>
         Monokino live performance in Paradiso, Amsterdam, 2005
      </title>
      <link href="http://stream.e5.ijs.si/publish/FAB_Channel_feed/20050526_monokino"
rel="alternate" type="application/xml"/>
      <updated>
         2011-04-04T16:37:40Z
      </updated>
      <id>
         urn:p2p-next:item:tag:stream.e5.ijs.si:/publish/FAB_Channel_feed/20050526_monokino/
      </id>
      <summary>
         Monokino whatever synopsis
      </summary>
      <p2pnext:image src="http://stream.e5.ijs.si/images/p2p-next-logo.jpg"/>
      <p2pnext:broadcastType>
         vod
      </p2pnext:broadcastType>
      <p2pnext:mediaUri>
```

```
         http://stream.e5.ijs.si/publish/torrents/20050526_monokino.tstream
      </p2pnext:mediaUri>
      <p2pnext:mediaDuration>
         PT00H22M03S
      </p2pnext:mediaDuration>
   </entry>
</feed>
```

From the feed output it can be seen how the export url parameter (-e) and absolute publishing link (-p), defined while creating the feed, are used. The feed id, item id and item images get the default treatment as is explained in the getfeed tool description.

3.2.6.4.3.4 Obtaining the discovery feed

The discovery feed, when created, will include the FAB channel feed as well, besides three other feeds:

```
  xyz:~/src/Next-Share:{1}> python JSI/ProviderToolbox/tools/discoveryfeed.py -t "Discovery
feed" -e http://stream.e5.ijs.si/discovery.xml -p "P2P-Next JSI/RTV Slovenia
(livinglab@e5.ijs.si)"

<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom" xmlns:p2pnext="urn:p2pnext:contentfeed:2009">
   <title>
      Discovery feed
   </title>
   <link href="http://stream.e5.ijs.si/discovery.xml" rel="self"/>
   <id>
      http://stream.e5.ijs.si/discovery.xml
   </id>
   <updated>
      2010-11-22T12:47:02Z
   </updated>
   <author>
      <name>
         P2P-Next JSI/RTV Slovenia Living Lab (livinglab@e5.ijs.si)
      </name>
   </author>
   <p2pnext:image src="http://stream.e5.ijs.si/images/p2p-next-logo.jpg"/>
   <entry>
      <title>
         RTV - Studio city
      </title>
            <link   href="http://stream.e5.ijs.si/feeds/studio_city.xml"   rel="alternate"
type="application/atom+xml"/>
      <updated>
         2010-11-22T12:47:02Z
      </updated>
      <id>
         http://stream.e5.ijs.si/feeds/studio_city.xml
      </id>
      <summary>
         Tedenska enourna oddaja magazinsko-informativnega formata, ki ob ponedeljkih zvečer
ob 21.uri v živo teče na drugem programu TV Slovenija, je postala kultna zaradi svojega
alternativnega in drznega pristopa k obdelavi različnih tem in dogodkov. Studio City se
začenja z zgoščenim in učinkovitim pregledom dogodkov preteklega tedna, nato se dotakne
notranjepolitičnih in svetovnih dogodkov, sodobne kulture, ne izogiba se niti družbeno
perečih tematik. Provokativno anketno vprašanje obdrži gledalce pred zasloni do konca oddaje,
ko se izpišejo rezultati t.i. ''mini referenduma''.
      </summary>
      <category schema="urn:service-type" term="tv"/>
      <p2pnext:image src="http://www.rtvslo.si/podcasts/mmc.png"/>
   </entry>
   <entry>
      <title>
         FAB Channel feed
      </title>
      <link href="http://stream.e5.ijs.si/feeds/fab.xml" rel="alternate"
type="application/atom+xml"/>
      <updated>
         2010-11-22T12:47:02Z
      </updated>
```

```
         <id>
            http://stream.e5.ijs.si/feeds/fab.xml
         </id>
         <summary>
            Concerts in Paradiso, Amsterdam
         </summary>
         <category schema="urn:service-type" term="tv"/>
         <p2pnext:image src="http://stream.e5.ijs.si/images/p2p-next-logo.jpg"/>
      </entry>
      <entry>
         <title>
            RTV - Zapisi iz Močvirja
         </title>
         <link href="http://stream.e5.ijs.si/feeds/zapisi_iz_mocvirja.xml" rel="alternate"
   type="application/atom+xml"/>
         <updated>
            2010-11-22T12:47:02Z
         </updated>
         <id>
            http://stream.e5.ijs.si/feeds/zapisi_iz_mocvirja.xml
         </id>
         <summary>
            Glosa Marka Radmiloviča, začinjena s prefinjenim smislom za humor, ki je enostavno
   ne smete preslišati!
         </summary>
         <category schema="urn:service-type" term="radio"/>
         <p2pnext:image src="http://www.rtvslo.si/podcasts/mmc.png"/>
      </entry>
      <entry>
         <title>
            Best of Today
         </title>
         <link href="http://stream.e5.ijs.si/feeds/best_of_today.xml" rel="alternate"
   type="application/atom+xml"/>
         <updated>
            2010-11-22T12:47:02Z
         </updated>
         <id>
            http://stream.e5.ijs.si/feeds/best_of_today.xml
         </id>
         <summary>
            Insight, analysis and expert debate as key policy makers are challenged on the
   latest news stories.
         </summary>
         <category schema="urn:service-type" term="radio"/>
         <p2pnext:image src="http://www.bbc.co.uk/radio/podcasts/today/assets/_300x300.jpg"/>
      </entry>
   </feed>
```

3.2.6.4.3.5 ClosedSwarm items

The current implementation (v3.0) can generate ClosedSwarm (CS) keys and corresponding torrent files if the closed swarm option (–cs) is specified on the command line when creating (adding) the item. The CS key is generated automatically. If the item is protected with CS the listing of the feed (-l option) or Json exports will specify the item's corresponding CS key file in its output. A miscellaneous option '–find-cs-key' with an argument of an item identifier can be used to find the CS keys of the item directly (the feed directory option '-d' needs to be specified for this).

3.2.6.4.4 Publisher

The publisher wraps the Next-Share core and publishes all the torrent files found in the torrent storage (specified using the EXPORT_TORRENT_DIR variable in default_settings.py) and shares the torrent files' related content.

The publisher updates itself regularly. The interval used can be tuned using the UPDATE_INTERVAL variable as specified in the settings. The default update interval of 60 seconds is intended for testing, for production the value should be longer, around a few minutes.

During updates the publisher finds new or removed content and reacts accordingly.

To run the publisher from the command line for testing, use:

```
xyz:~/src/Next-Share:{1}> python JSI/ProviderToolbox/tools/publisher.py
```

The publisher log threshold is set to DEBUG, so you can follow the debugging messages. The ProviderToolbox otherwise logs the messages through the operating system's syslog; the logging level there is set to INFO, so it is possible to follow the working of the toolbox.

For production usage, one could use for now the publisher script in the JSI/ProviderToolbox/bin/publisher directory. After running the process should be detached (disown?) from the terminal so the terminal closure wouldn't affect the running process.

3.2.6.4.4.1 Controlling the publisher

The publisher can be controlled through process signals, send to the publisher proces:

- kill -TERM 'pid of the publisher' will terminate the publisher
- kill -HUP 'pid of the publisher' will force a publisher update
- kill -USR1 'pid of the publisher' will cycle the log level of the publisher (but only the logging to the terminal)

When running in the terminal, CTR-C will shutdown the publisher gracefully.

3.2.7 Conclusion

The content provisioning process implementation covers most of the basic steps as discussed in the section 3.2.1. The covered steps are content acquisition (3.2.1.1), content storage (3.2.1.2), content injection (3.2.1.4) and content removal(3.2.1.7). The processes were extended in the last period with ingest process monitoring, but currently only supporting live streams. This work is covered in more depth in deliverable D8.1.2 [8]. Major improvements regarding the previous version of the ingest solution are the integration with the rich metadata tools and the provision of a complete metadata ingest chain. The toolbox implementation covers at the moment mostly the features required for straightforward implementations. The implementation design is fairly extensible as needed for integration and implementation of various existing and emerging *NextShare* requirements.

4 Content Adaptation

The task 5.2.3, content adaptation, aims to provide adaptation of scalable multimedia data within the *NextShare* system. To fully support scalable codecs, the scalability awareness needs to be integrated into the core of the *NextShare* system. This integration approach ensures that only those parts of the scalable bitstream which are needed for the desired quality are transmitted through the *NextShare* system. Although the integration of scalability into the *NextShare* systems aims to be codec-agnostic and to support a variety of scalable codecs, the main codec to be integrated is the *Scalable Video Coding (SVC)* [4] extension of the *H.264/MPEG-4 Advanced Video Coding (AVC)* standard. Thus, this chapter describes the *SVC*-integration into the *NextShare* system. However, all integration steps are performed in a codec-agnostic way, which enables the integration of other scalable codecs without needing to perform further changes to the *NextShare* core.

An important part of the SVC-integration process is the piece-picking algorithm. The algorithm decides which pieces to download from which peers at which time instance and tries to find the best trade-off between downloading all pieces in time and displaying the best-possible quality at each time instance.

The remainder of this chapter is organized as follows: First, a short summary of the SVC architecture from [11] is provided, as the architecture provides the basis for the piece-picking algorithm. Next, a description of the context-related metadata is provided, as these metadata are required to reflect the capabilities of the end-user terminals and the user's preferences. This ensures that the piece-picking algorithm does not try to download layers that cannot be processed by the end-user terminals. Finally, the piece-picking algorithm is described in detail. The piece-picking algorithm will be integrated into the *NextShare* system as joint work of WP4, WP5 and WP6 in the future.

4.1 SVC Architecture

This section summarizes the SVC architecture from [11], which has been jointly developed by WP4, WP5, and WP6. In the first part the selection of the scalability layers is described. Based on these layers, the mapping to the Bittorrent pieces is described in detail.

4.1.1 Scalability Layers

As a first step for the integration of *SVC* into the *NextShare* system, it was decided to provide four different *SVC* layers. The following layer strucuture is suggested for the first integration step:

Bitrate	**Resolution**	**Quality**	**frmps**
512 Kbps	320x240	low	25
1024 Kbps	320x240	high	25
1536 Kbps	640x480	low	25
3072 Kbps	640x480	high	25

Table 3: Scalability Layers

The decision for this layer structure has been taken based on several criteria. Firstly, these layers represent the commonly used bitrates for the videos distributed through the Internet today. Secondly, the bitrate between the layers increases smoothly, which enables good predictions for the

SVC encoding and ensures a low overhead in terms of bitrate for the scalability support. Thirdly, no HD layers are utilized for the first integration, as the upload bandwidth of the average P2P user is not high enough to provide support for streaming of such high bitrates. It should be noted that the base layer bitrate contains the bitrate for audio bitstream additionally to the video bitrate, as the audio is muxed to the base layer. Thus, from the 512 Kbps of the base layer 128 Kbps are utilized for audio content and the remaining bitrate is utilized for the video content. To ensure backwards compatibility with existing players, the *SVC* layers are provided in separate files. Thus, also a player without *SVC* support could still display the H.264/AVC-compatible base layer. The files could be named as follows:

- SWARMNAME.ts
- SWARMNAME_1024Kbps.dat
- SWARMNAME_1536Kbps.dat
- SWARMNAME_3072Kbps.dat

Based on these files, the user could decide to only download the base layer, if *SVC* is not supported or only a low quality bitstream is desired. If a higher quality bitstream is desired, one or more enhancement layers are downloaded in addition.

4.1.2 Mapping to Bittorrent Pieces

The second step of the integration process is the mapping of the scalable layers to *Bittorrent's* pieces. The mapping of the scalability layers to pieces has to be performed based on two major criteria. Firstly, the number of frames contained in one unit should be not too large; this ensures that if the network conditions change the switching to a lower or higher quality can be performed within a few seconds. Secondly, the number of frames mapped to one unit should be not too low, to ensure that the number of pieces and the corresponding overhead for the piece management does not get too high. Based on these criteria, a mapping of 64 frames, which represent 2.56 seconds of content at a framerate of 25 frmps, to one unit for the base layer has been chosen. Based on measurements to find the optimal piece size, we have decided that such a unit is mapped to three pieces in the *NextShare* system. The resulting mappings are presented below:

Layer	**Kb/time slot**	**Kbyte/time slot**	**pieces/time slot**
BL	512 Kbps * 2.56 ≈ 1.310	/ 8 ≈ 164 KByte	3 pieces à 55 KByte/time slot
EL1	1024 Kbps * 2.56 ≈ 2.621	/ 8 ≈ 328 KByte	6 pieces à 55 KByte/time slot (3 pieces in previous layers, 3 new pieces)
EL2	1536 Kbps * 2.56 ≈ 3.932	/ 8 ≈ 492 KByte	9 pieces à 55 KByte/time slot (6 pieces in previous layers, 3 new pieces)
EL3	3072 Kbps * 2.56 ≈ 7.864	/ 8 ≈ 983 KByte	18 pieces à 55 KByte/time slot (9 pieces in previous layers, 9 new pieces)

Table 4: Calculation of Piece Mapping

Please note that the small overhead in the piece mapping is utilized to compensate the small drifts of the constant bitrate encoding [12] of the SVC bitstreams. Based on the calculation illustrated above, a mapping to *Bittorrent's* pieces could be performed as shown in the following figure:

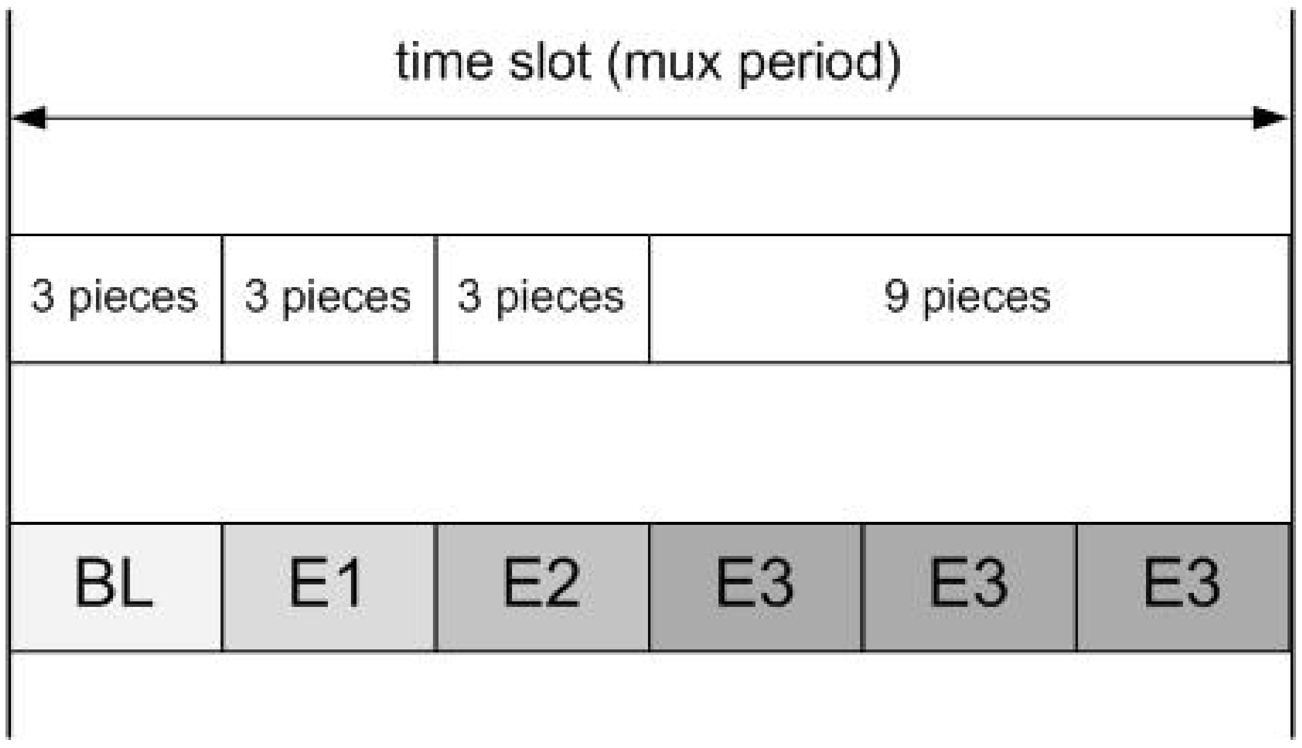


Figure 12: Piece Mapping

For the mapping shown above, the base layer is mapped to three pieces, the first enhancement layer to three pieces, the second enhancement layer to three pieces and the third enhancement layer to nine pieces. While the base layer is downloaded everytime, the decision which enhancement layers need to be downloaded is taken based on the available bandwidth provided by the other peers, the user preferences and the capabilites of the user's terminal as well as on the network capabilites and conditions. More details on this piece-picking are provided in Section 4.3.

Although a fixed piece size of 55 KByte has been suggested in this section, such a piece size might not always be the optimal choice, e.g., a lower or higher piece size might be preferable for better trading between peers. If another piece size is chosen, the piece mapping can still be used similar, i.e., by using n pieces for the base layer and 3n pieces for the third enhancement layer.

4.2 MPEG-21 DIA Context-related Metadata

Within the *NextShare* system the adaptation of the content is not only based on the network conditions but also on the user's environment. If the user is accessing the system with a mobile device that cannot display video content in high resolutions, it does not make sense to transfer the video in high quality to the user, even if the bandwidth is available. Thus, context-related metadata are utilized to ensure that the user only downloads those layers which can actually be displayed at the user's device. As context-related metadata, two tools from *MPEG-21 Part 7, Digital Item Adaptation (DIA)* [13], are utilized: The *Usage Environment Description* and the U*niversal Constraint Description*.

The *MPEG-21 DIA Usage Environment Description (UED)* allows the descriptions of the usage environment. This includes the description of terminal characteristics, network characteristics, user characteristics and the characteristics of the natural environment. An example for such a *UED* is given below:

```
<DIA xmlns="urn:mpeg:mpeg21:2003:01-DIA-NS" xmlns:dia="urn:mpeg:mpeg21:2003:01-DIA-NS"
xmlns:mpeg7="urn:mpeg:mpeg7:schema:2001" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
   <Description xsi:type="UsageEnvironmentType">
      <UsageEnvironmentProperty xsi:type="TerminalsType">
         <Terminal>
            <TerminalCapability xsi:type="CodecCapabilitiesType">
               <Decoding xsi:type="VideoCapabilitiesType">
                  <Format href="urn:mpeg:mpeg7:cs:VisualCodingFormatCS:2001:3.1"/>
                  <CodecParameter xsi:type="CodecParameterBitRateType">
                     <BitRate average="2000000" maximum="3000000"/>
                  </CodecParameter>
               </Decoding>
               <Decoding xsi:type="AudioCapabilitiesType">
                  <Format href="urn:mpeg:mpeg7:cs:AudioCodingFormatCS:2001:5.4.1"/>
                  <CodecParameter xsi:type="CodecParameterBitRateType">
                     <BitRate average="1280000" maximum="1560000" />
                  </CodecParameter>
               </Decoding>
            </TerminalCapability>
            <TerminalCapability xsi:type="DisplaysType">
               <Display xsi:type="DisplayType">
                  <DisplayCapability xsi:type="DisplayCapabilityType" colorCapable="true">
                     <Mode refreshRate="80">
                        <Resolution horizontal="1024" vertical="768"
                        activeResolution="true"/>
                     </Mode>
                     <ScreenSize horizontal="1024" vertical="768"/>
                     <ColorBitDepth blue="8" red="8" green="8"/>
                  </DisplayCapability>
               </Display>
            </TerminalCapability>
         </Terminal>
      </UsageEnvironmentProperty>
      <UsageEnvironmentProperty xsi:type="NetworksType">
         <Network xsi:type="NetworkType">
            <NetworkCharacteristic xsi:type="NetworkCapabilityType" maxCapacity="64000"/>
            <NetworkCharacteristic xsi:type="NetworkConditionType">
               <AvailableBandwidth minimum="1200" average="1200" maximum="1200"/>
               <Delay packetOneWay="50" packetTwoWay="100"/>
               <Error bitErrorRate="9" packetLossRate="0.0001"/>
            </NetworkCharacteristic>
         </Network>
      </UsageEnvironmentProperty>
   </Description>
</DIA>
```

The example *UED* shown above contains a description of the terminal and a description of the network. For the terminal, the codec capabilities for video and audio as well as the as the capabilities of the display are described. For the network, the capabilities as well as the current conditions are described. More details on the the properties of the *UED* are provided in [13].

The *MPEG-21 DIA Universal Constraint Description (UCD)* can be utilized to further constrain the usage of a Digital Item. Such constraints can be either defined by referencing values from the *UED* or by specifying numeric values for the constraints. An example for such a *UCD* is given below:

```
<DIA xmlns="urn:mpeg:mpeg21:2003:01-DIA-NS" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <DescriptionMetadata>
    <ClassificationSchemeAlias alias="SFO"
    href="urn:mpeg:mpeg21:2003:01-DIA-StackFunctionOperatorCS-NS"/>
    <ClassificationSchemeAlias alias="AQoS"
    href="urn:mpeg:mpeg21:2003:01-DIA-AdaptationQoSCS-NS"/>
    <ClassificationSchemeAlias alias="MEI"
    href="urn:mpeg:mpeg21:2003:01-DIA-MediaInformationCS-NS"/>
  </DescriptionMetadata>
  <Description xsi:type="UCDType">
    <AdaptationUnitConstraints>
      <LimitConstraint>
        <Argument xsi:type="SemanticalRefType" semantics=":MEI:17"/>
        <Argument xsi:type="ConstantDataType">
          <Constant xsi:type="IntegerType">
            <Value>352</Value>
          </Constant>
        </Argument>
        <Operation operator=":SFO:38"/>
      </LimitConstraint>
      <LimitConstraint>
        <Argument xsi:type="SemanticalRefType" semantics=":MEI:18"/>
        <Argument xsi:type="ConstantDataType">
          <Constant xsi:type="IntegerType">
            <Value>288</Value>
          </Constant>
        </Argument>
        <Operation operator=":SFO:38"/>
      </LimitConstraint>
      <LimitConstraint>
        <Argument xsi:type="SemanticalRefType" semantics=":MEI:9"/>
        <Argument xsi:type="SemanticalDataRefType" semantics=":AQoS:6.6.5.3"/>
        <Operation operator=":SFO:38"/>
      </LimitConstraint>
    </AdaptationUnitConstraints>
  </Description>
</DIA>
```

The *UCD* shown above contains three different limit constraints. Each limit constraint consists of two arguments and one operator. The first argument specifies the property that gets constrained, the second argument specifies the constraining value and the operator specifys the relationship between the two arguments, e.g., the first argument has to be lower than or equal to the second argument. The semantics of the arguments are speficied by utilizing the classification schemes defined in [13]. While the first two limit constraints use numeric values to constrain the resolution of the desired video to a width of *352* and a height of *288*, the third limit constraint defines that the bitrate of the video may not be larger than the average available bandwidth in the network, which is defined in the *UED*.

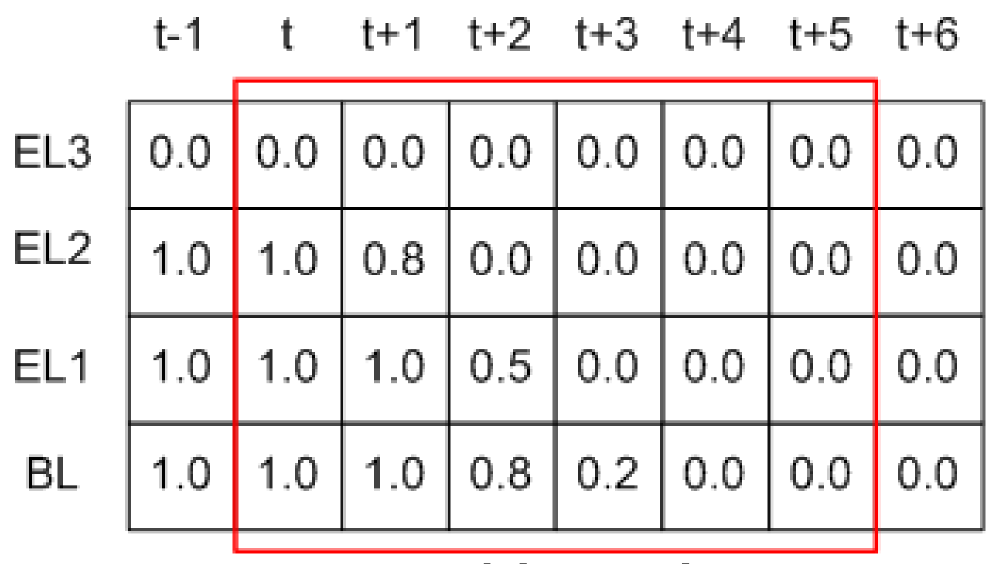
Utilizing the *UED* as well as the *UCD*, the capabilities of the user's terminal as well as the user preferences can be provided to piece-picking algorithm, which can subsequently select the maximum layer that is targeted for download based on this input.

4.3 Layered Piece-Picking

In the context of streaming layered video content over P2P networks, the piece-picking algorithm decides which pieces are downloaded at which time instance. The main goal of the algorithm is to find the best trade-off between displaying the best possible quality at every time instance and ensuring that all pieces are downloaded in time. When trying to download the best possible quality, the algorithm should not only ensure that the available bandwidth is utilized as extensively as possible, but should also avoid frequent switches in quality, as experimental measurements have shown that frequent quality switches are more disturbing for users than the constant playback of the video at lower quality [14]. Additionally, all pieces need to be downloaded before their display time, to avoid frame skipping or freezing of the video playback.

4.3.1 Introduction to Piece-Picking

The piece-picking decision has to be taken for all layers at all time instance. An illustration of such a decision process is provided below.



	t-1	t	t+1	t+2	t+3	t+4	t+5	t+6
EL3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
EL2	1.0	1.0	0.8	0.0	0.0	0.0	0.0	0.0
EL1	1.0	1.0	1.0	0.5	0.0	0.0	0.0	0.0
BL	1.0	1.0	1.0	0.8	0.2	0.0	0.0	0.0

Figure 13: Sliding Window

The piece-picking algorithm works on the sliding window illustrated above. In the sliding window, the value in each cell describes the download status for a piece which needs to be displayed at a specific time and belongs to a specific layer. At the actual time instance t, the algorithm has to decide which pieces to download for time points t+4 to t+5. In this situation, the algorithm might decide to download the enhancement layers for t+1 and t+2 to improve the quality for the near future or might decide to download the base layer for t+4 and t+5 to ensure that the playback will not stop, even if the network conditions become worse.

Although the sliding window only illustrates the download progress for each piece, there are five different piece states which are relevant for the piece-picking algorithm:

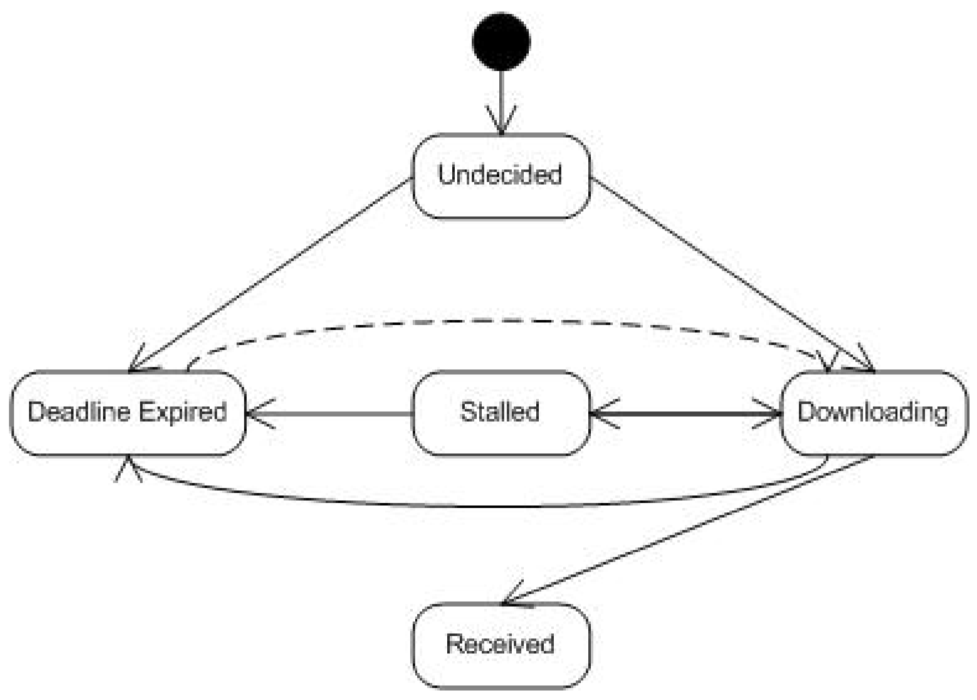


Figure 14: Piece-Picking States

The initial status for each piece is *undecided*. If the deadline of a piece expires, the status of the piece changes to *deadline expired*. On the other hand, if the piece-picking algorithm decides to start downloading a piece, its status changes to *downloading*. If the download of a piece is finalized, the status changes to *received*. Additionally, during downloading the status of a piece might be changed to *stalled* (and later *deadline expired*), if other pieces with higher priority require the available bandwidth. In case of suddenly improved network conditions an already expired piece might still be downloaded in order to provide them to other peers (but only if their is a huge amount of excessive bandwidth).

The piece-picking algorithm needs to decide on the pieces within the sliding window in frequent intervals. The frequency of these decision points is influenced by the size of the time slots as well as the arrival time of fully downloaded pieces. The possible intervals for the decision points are illustrated below:

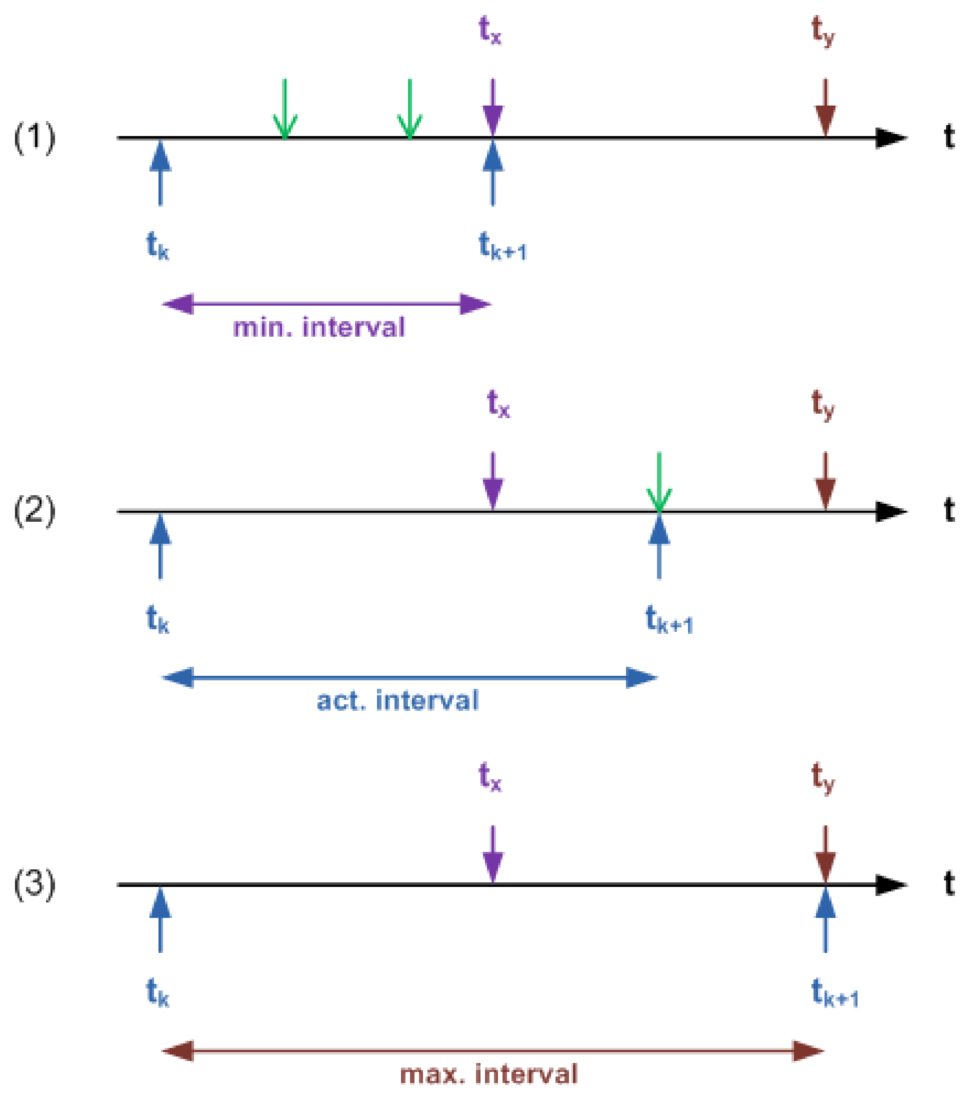


Figure 15: Decision Intervals

The decision points are separated by at least the minimum decision interval and at most the maximum decision interval. Thus, the decision can be taken based on one of the three possible situations:

1. If downloads are finalized during the minimum interval, the decision is taken as soon as the minimum interval is over.
2. If downloads are finalized between the minimum and the maximum interval, the decision is taken as soon as the download is finalized.
3. If no downloads are finalized during the maximum interval, a new decision is taken after the interval is over.

The three possibilities should ensure that not too many decision points are selected when downloads are finalized within very small gaps, but should also make sure that the decision is taken again if the downloads take longer than expected to be finalized.

In the following section, a description of the optimisation algorithm utilized to find the optimal piece selection is provided.

4.3.2 The Piece-Picking Algorithm

The piece-picking algorithm should provide the following output at every decision point:

- Which pieces to download or not to download.

- At which point in time to request the piece.
- From which neighbor peer to request a piece.

To find an optimal piece-picking algorithm, which maximizes the playback quality, ensures continuous playback, and avoids frequent switching of the playback quality, a number of algorithms have been developed and tested. The results have been published in papers [16][17][18]. A summary of the main findings is provided in the following.

The greedy piece-picking algorithm [16] was developed with the main goal to provide an algorithm which can adapt its behavior according to the current network conditions. The piece-picking decision of the greedy algorithm is taken with the following goals.

- Maximize the total utility of the bit stream, i.e., make sure that the best possible quality is downloaded while ensuring that the pieces are displayed in time.
- Minimize the changes in quality during the streaming process to provide a better experience for the user (see [14]).
- Minimize the number of bits transmitted to achieve a specific quality.

To achieve the desired output while following the optimization goals, the following three steps need to be performed: the piece utility calculation, the piece selection, and the peer selection. All three steps are described in detail in the following sections.

Piece Utility Calculation

During the piece utility calculation process, every piece within the sliding window is assigned a utility that is used to specify a order on the pieces. The utility is defined as

$$?_{???} = \frac{d_i \times w_{i\,j\,k}}{(t_j - t_k)^{\alpha}}$$

where d_i describes the distortion reduction of the piece, $w_{i\,j\,k}$ describes the probability to receive the piece in time, and the denominator describes the urgency of the piece (i.e., the difference between the time of the decision point, t_k, and the time when the piece needs to be displayed, t_j).

The weighted download probability is defined as

$$w_{i\,j\,k} = p_{i\,j\,k} \times w_{i-1\,j\,k} \times w_{i\,j-1\,k}$$

where $p_{i\,j\,k}$ describes the propability to receive the current piece in time, $w_{i-1\,j\,k}$ describes the weighted download probability for the piece of the lower layer (if there is one), and $w_{i\,j-1\,k}$ describes the weighted download propability for the same layer's piece of the previous time stamp, which should avoid frequent quality switches.

After the utility calculations, the pieces are ordered in two queues. First the pieces which are not being downloaded so far (status *undecided* or *stalled*) are ordered in a queue according to their utility and cost (i.e., the size of the piece), starting with the highest utility:

$$q_1 : \frac{U_1}{C_1} > \frac{U_2}{C_2} > \frac{U_3}{C_3} \cdots$$

Additionally, the pieces that are currently being downloaded are ordered in a second queue, according to their weighted download probability, starting with the lowest download probability:

$$q_2: w'_1 < w'_2 < w'_3 \ldots$$

The two queues are subsequently provided as input to the piece selection process.

Piece Selection

After the piece utility calculation, the pieces with the best utility need to be selected for download. The piece selection algorithm is specified as follows:

1. Select the y pieces with the lowest weighted download probability $w'_1, w'_2, \ldots, w'_y$ from the beginning of queue 2 and add them to queue 1.
2. Select the pieces with utility $U_1, U_2, \ldots, U_z$ from the beginning of queue 1, so that
   a) $C_1 + C_2 + \ldots + C_z - (C'_1 + C'_2 \ldots + C'_y) \leq B_k$
   b) $C_1 + C_2 + \ldots + C_z + C_{z+1} - (C'_1 + C'_2 \ldots + C'_y) > B_k$
   where B_k specifies the available free bandwidth at decision point t_k.
3. The utility $U_1 + U_2 \ldots + U_z - (U'_1 + U'_2 \ldots + U'_y)$ is the gain for this decision point.

The piece selection algorithm takes care of two problems: First, it should ensure that the most important pieces, i.e., those with the best distortion reduction, the highest probability to be downloaded in time, and which are needed most urgently, are downloaded first. Additionally, the download of pieces which are improbable to be received in time (e.g., because their deadline is very close and their is not sufficient bandwidth to download higher layer pieces) should be stopped and the gained bandwidth should be used to download more important pieces.

Peer Selection

After the pieces are selected for download, the peers from which the pieces should be downloaded need to be selected.

The peer selection algorithm is specified as follows:

1. Select a piece from the beginning of the list of selected pieces.
2. For each peer n_l in the list of neighbor peers, calculate the weighted download probability for the selected piece as well as the pieces being downloaded already from n_l to arrive in time if the selected piece is downloaded additionally from n_l. Select the peer with the highest average download probability as the best peer.
3. Repeat step 2 for the next piece in the list of selected pieces until the best peer for each piece has been selected.
4. If a peer has been selected as best peer by only one piece, the peer is assigned for download of the piece.
5. For the remaining pieces, the piece with the highest utility is assigned the selected peer for download, although it has been pre-selected by other pieces as well.
6. Remove the pieces with assigned download peers from the list and restart the algorithm at step 1.

The right choice of parameters for the piece-picking algorithm under various network conditions. have been evaluated through simulations using the Omnet++ [15] framework. Additionally, the greedy piece-picking algorithm has been compared to well-known mathematical algorithms which can be applied to the piece-picking problem and it has been shown that the greedy piece-picking algorithm can very well compete with the other algorithms, at clearly lower complexity [16].

To verify the results from the simulations using the greedy piece-picking algorithm, the algorithm has been integrated into *NextShare* and compared to other piece-picking algorithms for layered content [17]. The different piece-picking algorithms have been evaluated in a test lab with Linux machines. The server machines run P2P clients seeding the content and the client machines run P2P clients which are consuming the content. Between the machines running P2P clients, router machines are utilized to ensure the desired network conditions. The router machines use the Netem kernel component to emulate the network characteristics [19].

The overall results of these evaluations using *NextShare* confirm the results previously gathered by simulations. The greedy piece-picking algorithm provides a good performance at low complexity. Compared to previously published algorithms, the parameters of the greedy algorithm allow to adapt the algorithm's behavior to varying network conditions, which ensures that the performance over all tested scenarios is superior to the other evaluated algorithms. For detailed results of all test scenarios please refer to [17].

A further development of the greedy piece-picking algorithm, the deftpack algorithm, is presented in [18]. The deftpack algorithm uses the greedy piece-picking algorithm for the download of enhancement layer pieces, but always prioritizes base layer pieces over enhancement layer pieces. Furthermore, deftpack uses a dynamic sliding window size. The sliding window size is increased for peers with a slow network connection and decreased for peers with a fast network connection. The increase of the sliding window size for slow peers ensures that sufficient pieces of the base layer are buffered before switching to a higher quality. This is especially important in the case when neighbor peers leave the swarm and the download bandwidth of the peer is decreased for some time, as the buffered base layer pieces for multiple time slots ensure a continuous playback. For peers with a fast download bandwidth, the shorter sliding window ensures that the switch to the desired high playback quality occurs faster, as less base layer pieces are buffered.

This additional prioritization of the base layer and the dynamic sliding window size has shown to enhance the performance of the piece-picking algorithm in large-scale swarms (please refer to [18] for detailed evaluation results).

5 References

[1] D5.3.1, Michael Eberhard. Tools for rich metadata and signposts, P2P-Next project deliverable. December 2011.

[2] D5.4.1, Dominic Tinley. The LIMO tools for interactivity, P2P-Next project deliverable. December 2011.

[3] ISO/IEC 21000-2:2005, “Information Technology – Multimedia Framework – Part 2: Digital Item Declaration”, 2005.

[4] H. Schwarz, D. Marpe, T. Wiegand, "Overview of the Scalable Video Coding Extension of the H.264/AVC Standard", *IEEE Trans. on CSVT*, vol. 17, no. 9, September 2007, pp. 1103-1120.

[5] ISO/IEC FDIS 21000-9:2005: "MPEG-21 Part 9: File Format", January 2005.

[6] Cubewerx Binary XML library, http://www.cubewerx.com/bxml.

[7] D4.0.6, NextShare Platform M48, P2P-Next project deliverable, A. Bakker. December 2011.

[8] D8.1.2, Initial Analysis of the Initial live prototype implementation, P2P-Next project deliverable. June 2010.

[9] URIPlay, http://uriplay.org/.

[10]Wikipedia contributors, "Adobe Flash," *Wikipedia, The Free Encyclopedia*, http://en.wikipedia.org/w/index.php?title=Adobe_Flash&oldid=330714006.

[11] D6.5.5, N. Capovilla et al., NextShare Intermediate Integration v.6, P2P-Next project deliverable. September 2011.

[12] D.6.1.4, N. Capovilla. H.264/SVC codec v.4, P2P-Next project deliverable. December 2009.

[13] ISO/IEC 21000-7:2007, “Information Technology - Multimedia Framework - Part 7: Digital Item Adaptation”, 2007.

[14] M. Zink, O. Kuenzel, J. Schmitt, R. Steinmetz, „Subjective impression of variations in layer encoded videos“. IWQoS 2003, pp. 137-154.

[15] Omnet++, http://www.omnetpp.org/.

[16] Michael Eberhard, Tibor Szkaliczki, Hermann Hellwagner, László Szobonya, and Christian Timmerer. Knapsack problem-based piece-picking algorithms for layered content in peer-to-peer networks. In Proceedings of the 2010 ACM workshop on Advanced Video Streaming Techniques for Peer-to-Peer Networks and Social Net-working, AVSTP2P '10, pages 71#76, New York, NY, USA, 2010. ACM.

[17] Michael Eberhard, Riccardo Petrocco, Hermann Hellwagner, and Christian Timmerer. Comparison of piece-picking algorithms for layered video content in peer-to-peer networks. In Proceedings of the Consumer Communication & Networking Conference 2012, CCNC'12, to be published.

[18] Riccardo Petrocco, Michael Eberhard, Johan Pouwelse, and Dick Epema. Deftpack: A robust piece-picking algorithm for scalable video coding in P2P systems. In Proceedings of

the International Symposium on Multimedia 2011, ISM'11, to be published.

[19] Netem network emulation. http://www.linuxfoundation.org/collaborate/workgroups/networking/netem.

Annex A

A Uniform Resource Name (URN) Scheme for P2P-Next and its Digital Items

Status of this Memo

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

This Internet-Draft will expire on April 29, 2010.

Abstract

This document describes a Uniform Resource Name (URN) scheme some resources it produces or manages.

1. Introduction

The P2P-Next integrated project will build a next generation Peer-to-Peer (P2P) content delivery platform, to be designed, developed, and applied jointly by a consortium consisting of high-profile academic and industrial players with proven track records in innovation and commercial success.

For the content assets distributed using P2P-Next technology, P2P-Next would like to assign unique, permanent, and location-independent names based on URNs for some resources it produces or manages.

This URN scheme specification is a formal URN scheme.

2. Specification template

2.1. Introduction

This section specifies a template for an URN scheme to be used for resources produced and managed by P2P-Next. It comprises the following assets:

1. Base template
2. Namespace
3. Item for usage within dii:Identifier [M21DIIa][M21DIIb]
4. Type for usage within dii:Type [M21DIIa][M21DIIb]

Note: "dii" is a namespace prefix and resolves to urn:mpeg:mpeg21:2002:01-DII-NS as defined in [M21DIIa].

2.2. Base template

All other URNs derived from this base template inherit the properties as defined in this section.

URN ID:

"p2p-next"

Registration Information:

Version: 1
Date: 2001-11-20

Declared registrant of the URN scheme:

Name: Christian Timmerer
Title: Ass.-Prof. Dipl.-Ing. Dr.
Affiliation: Klagenfurt University
Address: Universitätsstrasse 65-67, A-9020 Klagenfurt
Phone: +43 (463) 2700-3621
Email: christian.timmerer@itec.uni-klu.c.at

Declaration of structure:

URNs used to distribute content using P2P-Next technology will have the following hierarchical structure

urn:p2p-next:{usage name}:{assigned US-ASCII string}

where "{usage name}" is a US-ASCII string that conforms to URN Syntax requirements ([RFC2141]) and corresponds to the usage (such as "item", "ns", "type", etc.) as defined in subsequent sections and "{assigned US-ASCII string}" is a US-ASCII string that conforms to URN Syntax requirements ([RFC2141]).

The individual URNs shall be assigned by P2P-Next through the process of development of P2P-Next deliverables.

Relevant ancillary documentation:

None

Identifier uniqueness considerations:

P2P-Next shall establish unique identifiers as appropriate.

Uniqueness is guaranteed as long as the assigned string is never reassigned for a given usage name and that the usage name is never reassigned.

Identifier persistence considerations:

P2P-Next is committed to maintaining the accessibility and persistence of all resources that are officially assigned URNs by the organization.

Persistence of identifiers is dependent upon suitable delegation of resolution at the level of "usage name"(s), and persistence of usage name assignment.

Process of identifier assignment:

Assignment is limited to the owner and those authorities that are specifically designated by the owner. P2P-Next may designate portions of its URN for assignment by other parties.

Process of identifier resolution:

The owner will develop and maintain "URN catalogs" that map all assigned URNs to Uniform

Resource Locators (URLs) specifically to enable Web-based resolution of named resources. In the future an interactive online resolution system may be developed to automate this process.

The owner will authorize additional resolution services as appropriate.

Rules for Lexical Equivalence:

The "usage name" is case-insensitive. Thus, the portion of the URN:

urn:p2p-next:{usage name}:

is case-insensitive for matches. The remainder of the identifier must be considered case-sensitive.

Conformance with URN Syntax:

No special considerations.

Validation mechanism:

None specified. The owner will develop and maintain URN catalogs.

The presence of a URN in a catalog indicates that it is valid.

Scope:

Global

2.3. Namespace

Usage ID:

"ns"

Declaration of structure:

Namespace definitions such as used within XML Schemas or other programming languages will have the following hierarchical structure

urn:p2p-next:ns:{assigned US-ASCII string}

where "{assigned US-ASCII string}" is a US-ASCII string that conforms to URN Syntax requirements ([RFC2141]) and corresponds to the actual namespace assigned.

The individual URNs shall be assigned by P2P-Next through the process of development of P2P-Next deliverables.

2.4. Item for usage within dii:Identifier

Usage ID:

"item"

Declaration of structure:

Item definitions for unique identification of Digital Items distributed using P2P-Next technology. This URN shall be used within MPEG-21 dii:Identifier elements and will have the following hierarchical structure

urn:p2p-next:item:{assigned US-ASCII string}

where "{assigned US-ASCII string}" is a US-ASCII string that conforms to URN Syntax requirements ([RFC2141]) and corresponds to the identifier of the actual resource.

The individual URNs shall be assigned by P2P-Next through the process of development of P2P-Next deliverables.

2.5. Type for usage within dii:Type

Usage ID:

"type"

Declaration of structure:

Type definitions for unique type identification of Digital Items distributed using P2P-Next technology (such as "item", "rm:core", "limo", "rm:payment", "signposts", etc.). This URN shall be used within MPEG-21 dii:Type elements and will have the following hierarchical structure

urn:p2p-next:type:{assigned US-ASCII string}

where "{assigned US-ASCII string}" is a US-ASCII string that conforms to URN Syntax requirements ([RFC2141]) and corresponds to the identifier of the actual resource.

The individual URNs shall be assigned by P2P-Next through the process of development of P2P-Next deliverables.

3. Examples

The following examples are not guaranteed to be real. They are presented for pedagogical reasons only.

```
urn:p2p-next:ns:metadata:2008
urn:p2p-next:item:bbc-bbcone-b00n9p5x
urn:p2p-next:type:item:2009
urn:p2p-next:type:rm:core:2009
urn:p2p-next:type:rm:payment:2009
urn:p2p-next:type:rm:scalability:2009
urn:p2p-next:type:limo:2009
```

4. Namespace considerations

TBD

5. Community considerations

TBD

6. Security considerations

There are no additional security considerations other than those normally associated with the use and resolution of URNs in general.

7. IANA considerations

The IANA has registered formal URN XX, to P2P-Next within the IANA registry of URN NIDs.

8. Normative references

[M21DIIa] ISO/IEC 21000-3:2003, Information technology -- Multimedia framework (MPEG-21) -- Part 3: Digital Item Identification, March 2003.

[M21DIIb] ISO/IEC 21000-3:2003/Amd 1:2007, Information technology -- Multimedia framework (MPEG-21) -- Part 3: Digital Item Identification, January 2007.

[RFC2141] Moats, R., "URN Syntax", RFC 2141, May 1997.

9. Author’s Address

Christian Timmerer
Klagenfurt University, Dept. of Information Technology

Universitätsstrasse 65-67
A-9020 Klagenfurt

Phone: +43 (463) 2700 3621
EMail: christian.timmerer@itec.uni-klu.ac.at

Michael Eberhard
Klagenfurt University, Dept. of Information Technology
Universitätsstrasse 65-67
A-9020 Klagenfurt

Phone: +43 (463) 2700 3627
EMail: michael.eberhard@itec.uni-klu.ac.at

Michael Grafl
Klagenfurt University, Dept. of Information Technology
Universitätsstrasse 65-67
A-9020 Klagenfurt

Phone: +43 (463) 2700 3600
EMail: mgrafl@edu.uni-klu.ac.at

10. Full Copyright Statement