Content Sharing Middleware for Mobile Devices

Petros Belimpasakis, Juha-Pekka Luoma and Mihaly Börzsei
Nokia Research Center
P.O.Box 1000, 33721
Tampere, Finland
{petros.belimpasakis, juha-pekka.luoma, mihaly.borzsei}@nokia.com

ABSTRACT
As mobile smart phones have evolved being converged devices, they are able to generate and consume different types of high quality content, such as images, music and video. The need of people to share, synchronize and archive their content, lead to the creation of multiple related mobile applications and adoption of existing protocols for content sharing. In this paper we describe selected protocols suited for content sharing (UPnP, Atom and WebDAV), and we present the concept of Mobile Content Sharing Middleware, which provides a common interface for applications to access the different underlying transfer protocols and bearers, in a way that the schematic differences of these protocols are hidden. A prototype implementation of the middleware was created for mobile devices based on Symbian OS.

Categories and Subject Descriptors
H.3.5 [**Information Storage and Retrieval**]: Online Information Services – *Data sharing, Web-based services.*

Keywords
Content, sharing, middleware, Symbian, mobile devices

1. INTRODUCTION
Advanced mobile devices, capable of combining many features and technologies, are nowadays becoming commonplace. The best examples of such converged devices are smart phones which include the functionality of multiple traditional devices, such as mobile phones, digital cameras, digital music players, personal navigation systems, etc. Especially, the disruption that smart phones with image capturing function caused is huge. Camera phones have already outstripped digital still camera sales by a ratio of more than 4:1 and it is expected to reach 7:1 by 2010 [15]. Thus, mobile phones are likely to be become the primary consumer device for creating digital content. Users are now capable of creating their own content, easily and fast, opening new opportunities for services and applications around the user-generated content.

Apart from storing their content, for later retrieval and viewing, the users also like to share it with other people, as an extension of the human need for communication and experience sharing. Sharing is a very wide term and it can be executed though many paradigms. From the simple scenario of sending an image, over Multimedia Messaging Service (MMS) from a mobile phone to another one, to more advanced use cases of sharing content from a smart phone to a large living room TV set, or uploading the content to web services and inviting others to fetch it.

Smart phones that use "open platforms" allow 3rd party developers to create their own applications and, via some pre-defined Application Programming Interfaces (APIs), allow these applications to access functions that the smart phone platform provides. For example, the smart phones that are based on the S60 [13] platform, which runs on top of Symbian [16] Operating System (OS) provide interfaces for 3rd party applications to easily send content over MMS, using the specified API. Such an application has to be aware of this API and should be developed with the specific sharing method in mind. If a new sharing method is added to the S60 platform, the end user applications would not be able to directly use it. The application developers would be required to study the newly added API, re-engineer their application, re-compile and finally re-deploy it to their customers that have already installed an older version. This is impractical and imposes limitations to the lifecycles of applications.

In this paper we present a specialized middleware, which we call *Sharing Middleware* that allows 3rd party developers to create applications with sharing capabilities, while being agnostic of the lower level sharing technologies, protocols and data transfer interfaces. The target is to allow developers to create applications with powerful sharing capabilities, but with limited programming complexity, while making their applications future-proof to new features and sharing technologies, by making an API which is stable, reliable and future-proof.

Some first steps towards this direction, on the Symbian platform, have already been done in [17], but the focus was on providing a remote storage framework that would mount remote storage servers on the local file system of a device. Our work is focusing more on the higher levels of the content sharing aspects, taking more into consideration metadata, thumbnails/preview icons, access rights, as well as aspects of proximity based sharing. We are designing a middleware to be used by metadata rich "media album like" instead of just "file manager like" applications.

In this paper, after briefly analyzing the most common sharing protocols (Section 2), we introduce our Sharing Middleware (Section 3), its design requirements and architecture. In Section 4 we present a prototype middleware client, and in Section 5 some basic performance results of our Symbian OS based implementation. In Section 6 we present some related work done in the area of mobile content sharing, and finally in Section 7 we outline directions for future work.

MOBILWARE 2008, February 13-15, Innsbruck, Austria

DOI 10.4108/ICST.MOBILWARE2008.2757

2. SHARING TECHNOLOGIES

Before starting the specification of our middleware, we studied the most popular content sharing methods, as well as their respective protocols and assumed them to be the principal technologies to be initially supported by our middleware design.

In the Internet domain, content sharing is nowadays heavily based on the "offering" paradigm. We consider *offering* as the paradigm of making content (i.e. media items) available face-to-face or remotely, for others. Content items are not transferred unless downloaded by the other party, and only a copy can be taken. Examples include offering of pictures on a website, or offering music files via peer-to-peer networks. In the first case, hosting space can be on a 3rd party server, like the very popular Flickr [9] service, for images. In the second case, content is directly offered from one of the owner's devices to other devices. That could be done among devices in physical proximity, or devices communicating remotely. We assume that communicating devices are using the Internet Protocol (IP), and thus studied transfer technologies which work on top of it. Our study identified three protocols as the most widely used ones.

2.1 UPnP Audio/Video (AV)

The UPnP Forum has specified protocols which enable electronics devices, to discover and use each other's services. The UPnP AV architecture [18] has as a goal the selection and controlled discovery of media content at home. It introduces three elements:

- Media Server, a device hosting and offering content for browse/download, while also accepting content uploads
- Media Renderer, a device that can render (i.e. "play") content offered by a Media Server
- Control Point, an entity that coordinates the communication between the Media Server and Media Renderer.

In UPnP AV, metadata is defined in the Content Directory specification [19], and includes fields from the Dublin Core (DC) [3] metadata and Digital Item Declaration Language (DIDL) Lite [5]. Apart from adding metadata information to the content items, it also enables browsing and automatic searching of items hosted in a Media Server. Hypertext Transfer Protocol (HTTP) is the mandatory media transfer protocol, but other protocols can be optionally supported. UPnP AV protocols are nowadays supported by many network enabled consumer electronics devices, like computers, smart phones, stereo systems and high end television sets [3].

2.2 IETF Atom

The Internet Engineering Task Force (IETF) Atom Publishing Format and Protocol (atompub) working group specifies protocols for editing, syndicating and archiving sources of episodic media content. Used mainly for news and blog sites, it follows the web feed paradigm that allows a client to follow changes and updates on a web site that it is interested in. The main specifications are:

- The Atom Syndication Format [11], which describes "feeds" consisting of "entries", each with extensible metadata. The description XML-based and incorporates fields similar to the DC. Atom feeds are retrieved using HTTP GET with the feed URL, and managed using the Atom Publishing Protocol.
- The Atom Publishing Protocol [8] is the editing protocol for Atom feeds. It is based on HTTP for managing communication between a server and a client, and supports the normal HTTP POST, GET, PUT, DELETE for the respective CRUD (Create, Read, Update, Delete) operations on Atom entries. It benefits from features and optimizations generally available with HTTP, such as authentication, encryption and improved scalability through caching.

Atom protocols are nowadays used by most web sites that offer frequently updated content, such as blogs, social network based services, online photo albums and news sites. On the client side, it is supported by many browsers, feed readers and online album sharing applications. The name Atom without qualification is used interchangeably for Atom Syndication Format as well as for Atom Publishing Protocol.

Really Simple Syndication (RSS) is an XML format similar to Atom Syndication Format and used for the same purpose, mainly differing in Atom protocols being more strictly defined and richer in functionality and detail than the older RSS [6]. In content sharing, the biggest advantage of Atom protocols compared to RSS is the support for both upload and download of content and its descriptions, while RSS supports download only.

2.3 IETF WebDAV

The IETF Web-based Distributed Authoring and Versioning (WebDAV) [7] protocol defines HTTP extensions to enable distributed web authoring tools to be broadly interoperable, while supporting user needs. The HTTP with the WebDAV extensions basically gives read-write access to the clients and it can be used as a remote file system for the Internet.

Some of the key features of the WebDAV include overwrite prevents (using exclusive and shared locks), metadata (properties) and copy/move remote files, as well as versioning [1] and access control lists [2]. WebDAV is supported by major operating systems by default, as well as web servers like the Apache HTTP.

2.4 Comparison

Comparing the three given protocols, we see many similarities and a few key differences, which are summarized on Table 1.

Table 1. Brief protocol comparison

	UPnP AV	**Atom**	**WebDAV**
Underlying protocol	HTTP	HTTP	HTTP
User authentication	Not supported	Any HTTP auth [12]	Any HTTP auth [12]
Server/peer addressing	Dynamic (requires discovery)	Static (IP address / DNS name)	Static (IP address / DNS name)
Media files specified in	Content Item	Entry	Document
Collections of media files	Container	Feed	Collection
Nested collections	Yes	Limited	Yes
Metadata	DC & DIDL	XML similar to DC, extensible	XML based, extensible

Our target was to use the common features of all the protocols, as much as possible, and for the not so compatible features find the

least common denominator. Those were taken into consideration in the architectural design of our Sharing Middleware.

3. SHARING MIDDLEWARE

3.1 Concept

Users have their content distributed in many different repositories/services, which are accessible via different protocols, such as the ones mentioned previously. Uploading content to those services would require that the client supports the needed transfer protocols and has all the essential configurations, such as server IP address, credentials, etc. In a simple example, we can assume a user that generates content with a mobile smart phone and would like to upload it to different devices/services, so that it can be further shared with others. Depending on the sharing situation and target device, there are many different protocols that should be supported. For example:

- Home environment: In the home environment consumer electronics devices usually support the UPnP AV protocols. Thus, sharing in the home would mean that the mobile client needs to upload content using UPnP AV, to a network attached storage device, or other consumer electronics device, and allow other users to access it from there.
- Internet domain: In the Internet domain, many online photo albums and blogs allow content upload using Atom. Thus, sharing content, via online services, with a group of friends would require support of Atom on the client.
- Enterprise domain: Distributed storage is widely supported over the WebDAV protocol. So, this is one more protocol that would be required in a mobile device, in order to make content sharing possible in this environment.

On top of that, we have to also consider the access bearer/interface to be utilized by each protocol, on the specific sharing use case. For example, UPnP can be used in the home, over the WLAN infrastructure, but can also be used in WLAN ad-hoc mode, between two devices in physical proximity. Internet based services can be accessed via WLAN Internet hot-spots, wherever there is coverage, as well as via cellular network, e.g. using General Packet Radio Service (GPRS).

Similarly, another client that would need to download content from these repositories also needs to implement these protocols, in order to be able to browse and fetch the media items. Figure 1 shows just a few examples of possible protocols / services used while sharing content between two end-point devices.

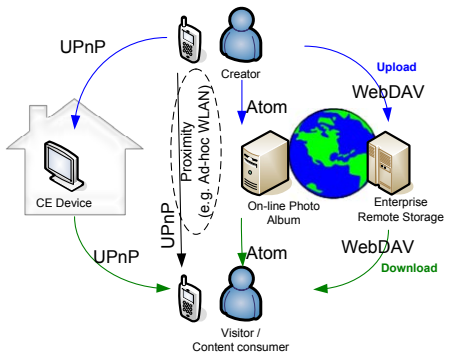

Figure 1. Example of content sharing protocols

A mobile application that would need to implement all these protocols would essentially become very complex. Adding the transfer protocol support in the mobile platform, as shared libraries, would allow multiple applications to use the available modules, but still it would require that applications are specifically aware of them, and have been designed with the knowledge of their respective APIs.

In our Sharing Middleware, we are introducing a new layer, between applications and content sharing protocol libraries, which provides a generic API to applications, for using any kind of underlying content transfer protocol, as shown in Figure 2.

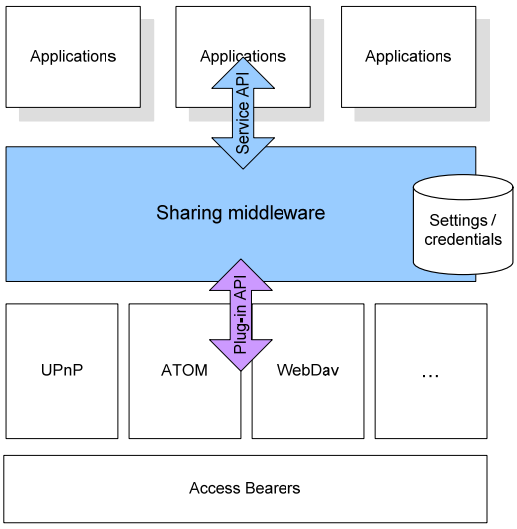

Figure 2. Sharing Middleware Simplified Design

The applications get from the Sharing Middleware a list of available content repositories, which we call *Sharing Accounts*, and simply decide to which one they want their items to be uploaded to. A sharing account, which is configured by the user, contains the required information for accessing a content repository, such as the location of the server, the authentication details, protocol to be used, etc. A simple upload request, by the application to the sharing middleware, along with the binary items, their respective metadata and target sharing account would be all that an application needs to do for uploading content, without being aware of the specifics of lower level transfer and transport protocols. Vice versa, another mobile device could request from its sharing middleware to browse and download content from a content repository, using any of the supported technologies.

3.2 Functional Requirements

A few key requirements were set for the design of our Sharing Middleware. The most important ones are briefly stated here.

The clients (high-level sharing applications) should be able to:

- Use the middleware API to send and retrieve media files and their associated metadata descriptions to/from a content repository.
- Use the middleware API to access the contents of a repository stored on a remote location accessible via the Internet, Local Area Network, or proximity network the device is connected to.

- Browse the contents of several Sharing Accounts combined as one, for aggregating different content repositories in a common view.
- Send the sharing middleware configuration parameters of a Sharing Account to another client (invitation), enabling the second client to access it.

The Sharing Middleware should be able to:

- Access Sharing Accounts which are implemented using server-based or serverless file transfer techniques.
- Handle temporary network discontinuation, and thus offline state handling between the device and a remote server. Reason is that under some cases, e.g. poor network coverage, mobile devices might not have continuous connection to the content repository.
- Extend its functionality by dynamically adding new sharing plug-ins and services as DLLs that are loaded automatically, when initializing the middleware.
- Provide interfaces, to the client applications, as C++ pure abstract classes which are easy to use.

3.3 Architecture

An overview of the Sharing Middleware architecture is shown in Figure 2, depicting the sharing services API that provides uniform access for a number of applications to a set of sharing mechanisms. Also shown is the plug-in API that is used to provide the sharing middleware with uniform access to a number of sharing protocols, each wrapped in dynamically registered plug-in. The plug-ins in turn utilize the data transport services provided by different access bearers.

3.3.1 Design Principles

The Sharing Middleware architecture was designed to be an extensible and flexible framework for both developing applications based on Sharing Middleware and for extending the middleware itself. The architecture should facilitate adding functionality and support for new protocols in the middleware aiming at compatibility with clients designed for earlier versions of the middleware. The class hierarchy for Sharing Middleware resulted from our comparison and analysis of different protocols suited for content sharing, followed by identifying the primary classes that needed to be represented in our framework, such as:

- item of shared content
- collection of shared content
- metadata of shared content
- address of shared content
- access rights (for shared content)
- sharing protocol (for content transfer)
- sharing service (client-side)
- sharing service provider (remote)
- sharing account (on a service provider)
- access credentials (for a sharing account)

While grouping the attributes and operations of each of the classes into a common base and its derived classes, the focus was on finding a small useful common subset for the base class. As an example, fields from the Dublin Core Metadata Element Set were the most common subset of metadata in our comparison and were thus included as the attributes of the metadata base class.

The middleware architecture includes a plug-in interface that enables DLLs implementing particular sharing protocols to be added to an installation of middleware without a need for recompilation. Two kinds of plug-ins are supported – *sharing plug-ins* and *sharing services*. These two differ in that a sharing plug-in is specific to a single sharing protocol, while a sharing service adds functionality on top of sharing plug-ins. For example, a sharing service could provide a local cache for content uploading, or in a more advanced case encapsulate automatic selection between sharing protocols and access bearers.

3.3.2 Top-down View

Figure 3 shows the two kinds of primary resources managed by the Sharing Middleware, sharing plug-ins and sharing services, deployed as plug-in DLLs. Symbian OS has a plug-in mechanism/framework which we use to make a sharing service and plug-in available to applications. For each such DLL detected, a sharing plug-in object (implementing the interface MSharingPlugin) and a sharing service object (implementing the interface MSharingService) are registered to the RSharingRoot object and available to applications via the respective methods getServices() and getPlugins().

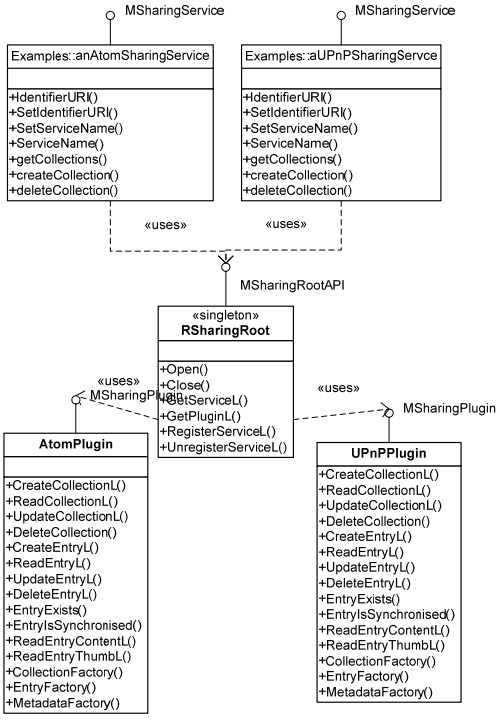

Figure 3. Relations of sharing root, services and plug-ins

3.3.3 Sharing Plug-ins

A sharing plug-in object is used as a wrapper to an implementation of a particular sharing protocol, such as Atom or UPnP. The methods of a sharing plug-in object provide a set of generic CRUD operations on a remote media store accessible by multiple clients. For each sharing plug-in object, a corresponding sharing service object is accessible via the SharingRoot object.

The clients of a sharing plug-in are the objects that constitute the sharing middleware data model (see 3.3.5) representing shared media files and collections. These objects, used by applications

via the sharing API services interface, utilize the services of a sharing plug-in to synchronize their internal state with that of a remote media store accessible via the sharing plug-in.

Figure 4 shows the interfaces derived from MSharingPlugin and inherited by concrete plug-in objects providing wrappers for concrete sharing protocol implementations (for Atom and UPnP protocols only in the diagram).

All sharing plug-in objects implement the interface class MSharingPlugin, which provides a common subset of the functionality provided by all sharing plug-ins. Finally, an interface class is defined for implementations of each protocol used for media sharing. This allows advanced applications needing the features particular to a certain sharing protocol and not available through the common API operations to use features specific to that protocol.

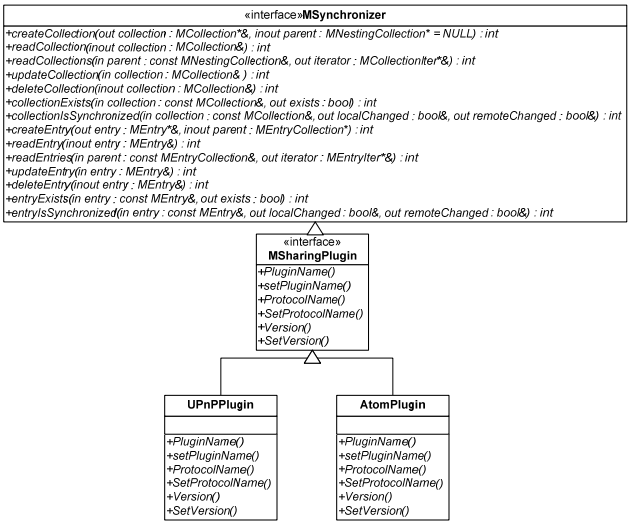

Figure 4. MSharingPlugin and derived classes

Meanwhile, providing different layers of specialization for different protocol plug-ins, of the sharing middleware, allows applications utilizing them to ignore the specifics of each sharing protocol as much as possible, yet able to distinguish between characteristics or instances of sharing protocols. By always using the most generic sharing middleware interface appropriate, applications can be made as future-proof as possible with additional plug-ins and new versions of the middleware, as support for new sharing technologies is added.

For plug-ins that implement the MDiscoverableSharingPlugin interface, the set of resources available for sharing may be different from time to time and takes some time to update. The device discovery is performed transparently and that takes place in the background.

3.3.4 Sharing Services

A sharing service object is used to manage collections of media entries accessible via a particular sharing plug-in. The client of each sharing service object is a sharing application using it to access a set of shared media collections.

All sharing service objects implement the interface MSharingService, and may additionally implement other interfaces derived from it. The set of available media collections and media entries available via a sharing service that implements the interface MDiscoverableSharingService is likely to change from time to time, for example as a result of the client device entering a LAN with a UPnP CDS compatible media server.

3.3.5 Data Model

The data model objects are used to represent sharing items and supplementary information in the memory of the device. Any changes made to a sharing item via a data model object are local only, until the method Update() is used to commit the changes by transferring the state of the local copy to a remote server/peer. The method Read() can be used similarly to transfer the state of the remote representation of a sharing item to the local copy of the sharing item.

Figure 5 shows relations between the classes that constitute the data model used for accessing media content via the sharing middleware. Each set of shared media files having access credentials and a set of metadata fields in common is represented as a collection object implementing the interface MCollection. A collection object may contain any number of sharing items, each implementing the interface MSharingItem.

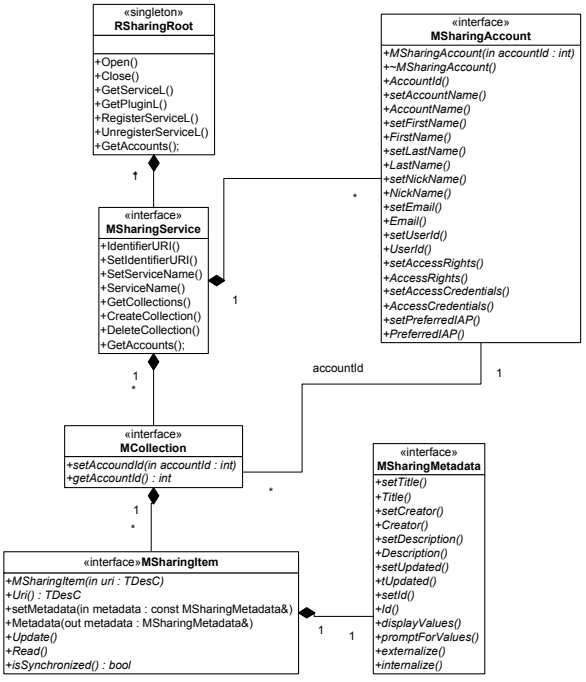

Figure 5. Relations between sharing root, services and items

The data model was designed to handle cache related items in controlled way. There are two interface classes available for clients to interact with the middleware, the MSharingPlugin and MSharingService. The plug-in interface does not keep the track of the downloaded data. It is the MSharingService that maintains a cache, however due to data traffic cost, on mobile networks, data models are only updated upon client request. The cache contains the Uniform Resource Identifier (URI) to the content item and its metadata only after a *Synchronization* of a collection. The media items are cached once they have been requested for download.

Collections
A collection object can be a container for media entries, other collections or both media entries and collections. This depends on whether a particular collection object implements one or both of the interfaces MEntryCollection and MNestingCollection.

- The UPnPCollection object corresponds to a container object in a UPnP Content Directory Service [19]. Because containers in UPnP CDS may include media items and other containers, interfaces MNestingCollection and MEntryCollection are both inherited by UPnPCollection.

- The AtomCollection object corresponds to an Atom feed. An Atom feed is defined in [11] to contain only media entries as its subelements. Thus, AtomCollection is a collection object implementing only the MEntryCollection interface.

- The AtomWorkspace object corresponds to an Atom workspace defined in [8]. Since a workspace in Atom is defined to contain only Atom feeds, the AtomWorkspace is a collection object implementing only the interface MNestingCollection.

A UPnP sharing service object acts as a container of the UPnP collections registered to sharing middleware and available to applications. Each UPnP collection in turn contains media entries and/or nested UPnP collections. Similarly, Atom sharing service object is a container for Atom collections, each containing entries.

Entries
Entries are the objects implementing the interface MEntry. Each of these objects represents a media entry, consisting of media content (image/video file) and the metadata of the content.

Metadata
The metadata fields used in the Sharing Middleware API are a subset of the Dublin Core Metadata Element Set. The format of the textual metadata fields used with this API is as defined for fields of the same name in the Atom Syndication Format [11].

Sharing Accounts
The details of an account on a sharing service are stored in an object where the account is associated with a set of access rights and optionally with a set of access credentials. Access rights describe the permissions that the user has on a sharing service. Access credentials provide the information needed by a sharing server/peer to authenticate the user.

3.3.6 Asynchronous Operations
Many of the operations in the sharing middleware API involve transport or reception of data between the client and a remote server/peer with relatively long delays, possibility of failure and in some cases incremental completion. There are classes defined to support this pattern of deferred return values. Active Objects are used internally by the Symbian C++ implementation of the sharing middleware to implement the deferred operation. The results of a deferred object are reported via different callbacks, and additionally incremental progress may be reported.

3.4 Interface Example
An example code of how applications could use the middleware, for downloading and uploading content is given here:

```
// Open a session to the SharingRoot
RSharingRoot* iSharingRoot=RSharingRoot::OpenL();

// Retrieve the list of sharing accounts
RAccountArray Accounts=iSharingRoot->GetAccounts();
// Select a service based on the account
MSharingService* SharingService=iSharingRoot-
>GetPluginL(*Accounts[iUserSelectedAccount])

// Create an empty collection
SharingService->CollectionFactory(iCollection);
// Assign the Account with the collection
Collection->SetAccount(*Accounts[iUserSelectedAccount]); //
Assign the SharingService and SharingRoot with collection
Collection->SetSynchroniser(SharingService,SharingRoot);

// Optional - For upload an entry with atom - start
SharingService->EntryFactory(DLEntry); // Create empty entry
SharingService->FileContentFactoryL(FileContent); // Create a
File content descriptor
FileContent->SetFileNameL(KFilename()); // File descriptor
FileContent->SetFilePathL(KFolder()); // Setup the file descriptor
DLEntry->SetContent( *FileContent);  //Add FileContent to Entry
(static_cast<MEntryCollection*>(Collection))-
>AddEntry(*DLEntry); // Add the Entry to the Collection
// Optional - For upload an entry with atom - end

// Set the (MDeferredObserver) as observer for the sharing events
Collection->Synchronize(this);
```

The optional part is used for uploading content to a remote server, when synchronization is done. If it is omitted the middleware will only download the list of entries from the remote server.

4. CONCEPT CLIENT OF THE MIDDLEWARE
Our sharing middleware has been designed to be used by applications that would require access to different kinds of sharing and transport technologies. It provides limited user interfaces and assumes that media view is handled by the client applications, which uses the functions of the middleware. Here we present some concepts of how such an application could look like, for accessing content that is distributed on different devices.

First of all, the middleware would need to have information about the albums that the user has access to. This configuration could be done manually, via a user interface (UI), or automatically by receiving a configuration message containing the required settings, e.g. over SMS. In the first case, the user would need to enter to the application, once, the settings and parameters of the accounts that he/she has access to. As shown in Figure 6, each account is linked to a plug-in type and depending on this the rest of the settings are dynamic.

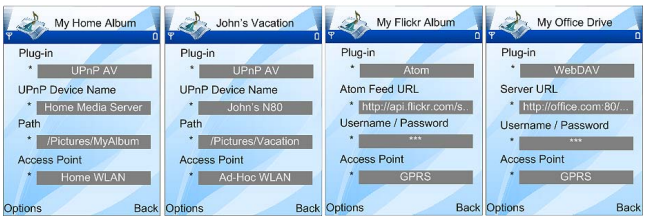

Figure 6. Concept UI for account configuration

For example, in the case of UPnP (or any other plug-in implementing the MDiscoverableSharingPlugin, as mentioned in 3.3.3) the discovery name of the target device is required, potential path to the shared collection/container and the interface

(called "Access Point" in S60), through which the connection should be established.

In the case of an Atom or a WebDAV account, the URL to the server path, as well as the credentials (username / password) are required, along with the Access Point. Figure 6 presents examples of accounts located a) on a home UPnP Media Server, accessed via home WLAN network, b) on another mobile UPnP device, accessed via WLAN ad-hoc connection, c) on an Atom-based Internet photo album, accessed via cellular GPRS, and d) on a WebDAV remote storage, also accessed via cellular GPRS.

When the application needs to share, by uploading, content to a repository, it can just query the user to select the destination, by giving a list of the preconfigured accounts, as shown in Figure 7.

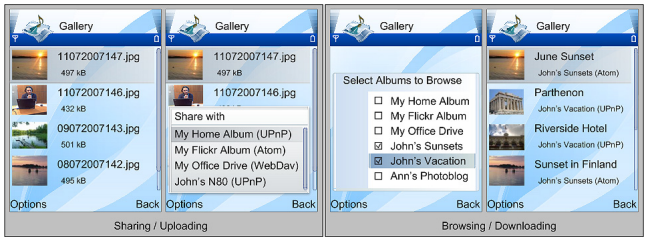

Figure 7. Concept UI for uploading & downloading content

In a similar manner, browsing and downloading content can start by allowing the user to select the albums to be accessed. It is possible to request fetching content from multiple albums, no matter where it is really located, that would appear in a common aggregated view. In the first phase, the application would request from the middleware only the metadata and thumbnails of the items, thus keeping data transfers to the minimum, for low latency and potential data transfer cost. Only if the user requests to further download an item, the full binary would need to be retrieved.

5. PERFORMACE OF IMPLEMENTATION

We implemented the Sharing Middleware for Symbian OS, on top of the S60 v.3.0 platform [13], and tested it on the Nokia N80 device. This specific device is equipped both with a UPnP library and an Atom library. We had full access to the device's UPnP stack and interface which, however, are currently not accessible via the public Software Development Kit (SDK). Our implementation work was focused on:

- The core of the Sharing Middleware.
- The creation of wrappers around the existing UPnP/Atom libraries to work as Sharing Middleware plug-ins.
- The development of a user interface to be used as a prototype application accessing the Middleware APIs.

On the server side, we used existing servers, such as:

- The UPnP PC server software provided with the N80 device. We run it on top of a Windows XP based laptop.
- The Flickr [9] online sharing service, which supports the Atom protocols.

For testing the performance of our implementation we did measurements by uploading media files, from the Nokia N80 device, to both repositories (the UPnP server and the Atom enabled Flickr service). In our setup, the laptop and the N80 smart phone were connected on a WLAN access point, supporting the IEEE 802.11g standard, which was then connected to the Internet, via a Symmetric DSL connection of 4Mbps speed.

We transferred, both to Flickr and to the home UPnP AV server, JPEG images of different sizes, in bundles of 1, 5 and 10 items. For each bundle, the total size of the contents was measured with volume of 500kB, 1000kB, 1500kB, 2000kB, 2500kB and 3000kB. This means that we had 3 x 6 = 18 cases. We measured the total time duration of the transfers in each case. Every case was executed three times and the mean value of these executions was used. The final results are shown in Figure 8.

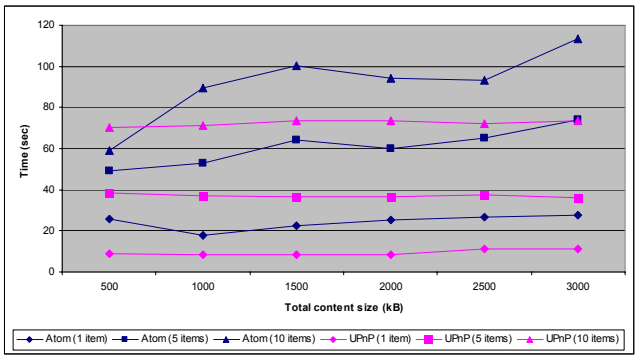

Figure 8. Total upload times, using Atom & UPnP

These initial results show that the total size of the content items does not really affect the transfer times. The reason is that we have been transferring very low data volumes, while the bandwidth available to the device is very high. What seems to heavily affect the total transfer time is the amount of items transferred, both in the cases of Atom and UPnP. In the worst case shown, uploading to Flickr one single file of 3000kB takes only 25 seconds, while uploading 10 files of the same total size takes 5 times more. This is something typical in on-line photo sharing web-sites, as they are generating multiple thumbnails of the uploaded images on-the-fly. We measured similar differences, i.e. up to 5 times longer upload times, when executing similar tests on a personal computer, with a Windows software client for Flickr.

UPnP transfers are clearly faster, since they are not affected by external factors (like thumbnail generation on the Internet services), and they do not require user authentication in the protected home domain.

6. RELATED WORK

Standalone mobile content sharing applications, for different environment and use case, have been studied and created by many researchers. The work of Sarvas et al. [14] created a system for online picture sharing, based on a client-server architecture, where the client (Symbian application) was designed to work with the dedicated server, over the HTTP protocol. Focus was on content sharing among social groups and friends, assuming that all content is gathered on a central, Internet-hosted, entity.

Tolvanen et al. [17] studied the support for remote storage clients on mobile devices. The focus of the work was on how remote servers could be mounted on the local file system, of a mobile Symbian device, for providing transparent file access to higher level applications. Efficient metadata handling was assumed to be done by some higher level applications and was not a key area of the work. Their implementation delivered a remote storage framework using the WebDAV protocol, as well as an enhanced

file browser. Based on this work, a WebDAV plug-in could be easily created for our Sharing Middleware.

Peer-to-peer content sharing has also entered the mobile environment, with multiple implementations trying to provide either clients for existing peer-to-peer networks, or utilizing other protocols. As an example, in [10] the authors demonstrate the usage of the Session Initiation Protocol (SIP) for creating a mobile peer-to-peer system, running on Symbian-based clients. With the appropriate wrappers, their implementation could also be used as a sharing plug-in, under our generic middleware.

7. FUTURE WORK

At this phase our Sharing Middleware provides interfaces for applications to access media repositories, as long as the appropriate plug-in exists and the configurations have been entered in the middleware. The selection of the sharing account, to be used is now done manually, most probably by the user. Our next step would be to enhance the middleware with the appropriate knowledge to make selections automatically. Feeding the middleware with the rich context information that a mobile device can have (from sensors, positioning systems, proximity interfaces, etc.) would allow it to suggest the most appropriate sharing account to the application.

Moreover, we would like to make the sharing experience much more user centric. Providing to the middleware the information that a mobile phone has in its contacts book, we would try to hide the content repositories, from the users, and show people instead. For example, user A could just select, from the contacts list, the name of the person (user B) that a video file should be shared with. The middleware should then try to identify what is the best way the content can be delivered to user B, taking in consideration the available devices, their capabilities, the size of the object to be transferred and available networks.

Meanwhile, we would like to get feedback on how useful this middleware is for other developers and allow them to build content sharing applications. So, we are currently studying the portability of the middleware to the latest S60 v.3.2 platform and potential public release as open-source.

8. CONCLUSIONS

In this paper we presented the design of a generic content Sharing Middleware, for mobile devices, that allows applications to transfer and access content through a generic interface. The interface was specified after studying the most common, IP based, sharing protocols, and identifying their common aspects, as well as normalizing the differences. A proof-of-concept prototype was built on the Symbian OS, on top of the S60 platform, along with two sample plug-ins, for UPnP and Atom protocols, that were tested against existing commercial devices and services.

The Sharing Middleware will eventually make the development of applications that utilize content sharing capabilities much easier and faster for the developers, who will not have to deal with the lower layer protocols. Moreover, support for other transfer protocols can be added, by creating plug-ins to be loaded dynamically, allowing further extension of the middleware. That means that content sharing, from mobile devices, could become simpler and more intuitive.

9. ACKNOWLEDGEMENTS

The authors would like to thank Seamus Moloney, from Nokia Research Center, and Prof. Jarmo Harju, from Tampere University of Technology for critically reviewing this manuscript.

10. REFERENCES

[1] Clemm, G., Amsden, J., Ellison, T., Kaler, C., Whitehead, J. Versioning Extensions to WebDAV , IETF, March 2002.

[2] Clemm, G., Reschke, J., Sedlar, E., Whitehead, J. WebDAV Access Control Protocol, RFC 3744, IETF, May 2004.

[3] DLNA compliant devices - http://product.dlna.org/eng/browse_cat.aspx (2007)

[4] Dublin Core Metadata Initiative, DCMI Metadata Terms, August 2006.

[5] Iverson, V., Song, Y.-W., Van de Walle, R., Rowe, M., Doim Chang, Santos, E. and Schwartz, T. MPEG-21 Digital Item Declaration, International Organization for Standardization, 2000.

[6] Comparison of Atom and RSS - http://intertwingly.net/moin-1.2.1/wiki/cgi-bin/moin.cgi/Rss20AndAtom10Compared

[7] Goland, Y., Whitehead, E., Faizi, A., Carter, S., Jensen, O. HTTP Extensions for Distributed Authoring -- WEBDAV, RFC 2518, IETF, February 1999.

[8] Gregorio, J., de hOra, B. The Atom Publishing Protocol, IETF Draft, IETF, December 2006.

[9] Flickr – http://www.flickr.com

[10] Matuszewski, M., Beijar, N., Lehtinen, J., Hyyrylainen, T. Mobile peer-to-peer content sharing application. Proceedings of the 3rd Consumer Communications and Networking Conference (CCNC 2006), 2006.

[11] Nottingham, M., Sayre, R. The Atom Syndication Format, RFC 4287, IETF, December 2005.

[12] J. Franks et al., HTTP Authentication: Basic and Digest Access Authentication, RFC 2617, IETF, June 1999

[13] S60 Platform – http://www.s60.com

[14] Sarvas, R., Viikari, M., Pesonen, J. and Nevanlinna, H. MobShare: Controlled and Immediate Sharing of Mobile Images. Proceedings of the 12th annual ACM international conference on Multimedia. October 10-16, 2004, NY, USA.

[15] Strategy Analytics: Camera Phone Sales Surge to 257 Million Units Worldwide in 2004, April 14th 2005.

[16] Symbian OS – http://www.symbian.com

[17] Tolvanen, J., Suihko, T., Lipasti, J., Asokan, N. Remote Storage for Mobile Devices. In Proceedings of the First International Conference on Communication System Software and Middleware, 2006.

[18] UPnP Forum, UPnP AV Architecture:0.83, June 2002.

[19] UPnP Forum,. ContentDirectory:1 Service Template Version 1.01, June 2002