Towards the UPnP-UP: Enabling User Profile to Support Customized Services in UPnP Networks

Thiago B. M. de Sales, Leandro M. de Sales, Marcos Pereira,
Hyggo Almeida, Angelo Perkusich, Kyller Gorgônio
Federal University of Campina Grande
Embedded Systems and Pervasive Computing Lab
Postal code 10105, ZIP 58109-970 - Campina Grande - PB - Brazil
{thiagobruno,leandro,marcos,hyggo,perkusic,kyller}@embedded.ufcg.edu.br

Marcello A. de Sales Jr.
San Francisco State University
Department of Computer Science msales@sfsu.edu

Abstract

In a nutshell, UPnP aims to enable simple and robust connectivity among consumer electronics, intelligent appliances and mobile devices from many different vendors. Nowadays, it is getting an increasing popularity due to its robust way of connecting devices and the massive number of already developed applications, complementing the useful new era of pervasive computing. Although UPnP is an easy-to-use solution for discovery, which allows controlled devices to share and access users contents, it does neither provide user authentication nor authorization mechanisms, requirements that would require user's data on personal profiles. As a consequence, we propose an extension of the UPnP protocol called UPnP-UP, which allows user authentication and authorization for UPnP devices and applications. The primary goal of our proposed extension is to modify the current UPnP specification as less as possible, maintaining backwards compatibility with previous versions of the protocol. Furthermore, we present our efforts on the development of a UPnP framework for publishing and accessing multimedia contents in order to support this extension as a case of study.

1. Introduction

With the advent of many personal mobile products like PDAs, cell phones and Internet tablets, the concepts of mobile and pervasive computing is becoming more and more practical. In this context, the UPnP standard [4] is a good candidate to provide pervasive services because it includes, among other features, zero-effort configuration and auto-discovering of devices. Similarly, due to the use of consolidated technology-based protocols for Internet applications such as HTTP, XML [2] and SOAP [10], UPnP has become the de-facto solution for personal devices and electronics gadgets, which enables pervasive computing through the sharing, access and interchange of user contents. As a matter of fact, it has been used in many open-source and proprietary projects throughout the Internet. One worth-citing is Canola [11], a mobile UPnP multimedia player for the Maemo Platform [3] developed by Nokia Institute of Technology in Brazil.

Besides the strong popularity and growth of manufactured UPnP-compatible devices, the UPnP standard doesn't provide user authentication or authorization mechanisms, limiting the development of user customized solutions such as music recommendation while using a media server or saving the state of the home devices, features that would follow certain form of user profile. In this way, this paper presents UPnP-UP, an extension of the UPnP protocol that enables user authentication and authorization for UPnP devices and applications. Although UPnP-UP provides authorization mechanisms, we will only present the authentication feature in this paper. Moreover, the goal of our extension is to modify the current UPnP specification as less as possible, maintaining backwards compatibility with the current UPnP implementations.

The remainder of this paper is organized as follows: After presenting some characteristics of the UPnP standard in Section 2, we detail the specification of the UPnP-UP in Section 3. Similarly, in Section 4 we present a case study where a UPnP multimedia system recommends songs to users according to their profile, presenting the state of UPnP-UP implementation. Finally we conclude with experiments results in Section 5, future works and the conclusions of this paper in Sections 6 and 7, respectively.

2. Overview and Background

The UPnP protocol first uses the control point [4], as the first step to *discover* UPnP-enabled devices, which may be filtered according to the user preference or the user's wish device type. The discovery process enables *description*, where the control point learns about the devices capabilities. In order to send commands to designated devices, the control point uses the step of *control*. In the *eventing* step, the control point keeps listening to the changes of state of the "hooked up devices", updating the graphical user inter-

face accordingly, which is defined in the *presentation* step.

When it comes to the discovery process, the control point sends a multicast message of type *M-SEARCH* [4], making it able to discover which UPnP devices are online. The remote devices running UPnP discovery protocol advertise their status by sending a unicast *NOTIFY* message back to the control point. An example of this message is shown in Listening 1. In that message, the device specifies essential information about it and a pointer – a URL for an XML file – that enable the control point to proceed with the following UPnP steps described above.

```
1 NOTIFY * HTTP/1.1
2 HOST: 239.255.255.250:1900
3 CACHE-CONTROL: <max-age>
4 LOCATION: <url of device description>
5 NT: <search target>
```

Listening 1: The UPnP NOTIFY message.

Through the URL listed on the *LOCATION* field (line 4), the control point has more information about the devices and pointer for services invocation through SOAP. The device description URL points to an XML instance, whose XML default schema defines the device's properties such as name, version and model, URLs for the devices' services directory, URLs to sign in for device events, among other information.

One of the most popular example that helps understanding the UPnP device is the interaction between UPnP A/V devices [9], shown in Figure 1. It is a schema of that describes the functionalities used by the a control point to browse multimedia items from a media server and how to present these items into the media renderer.

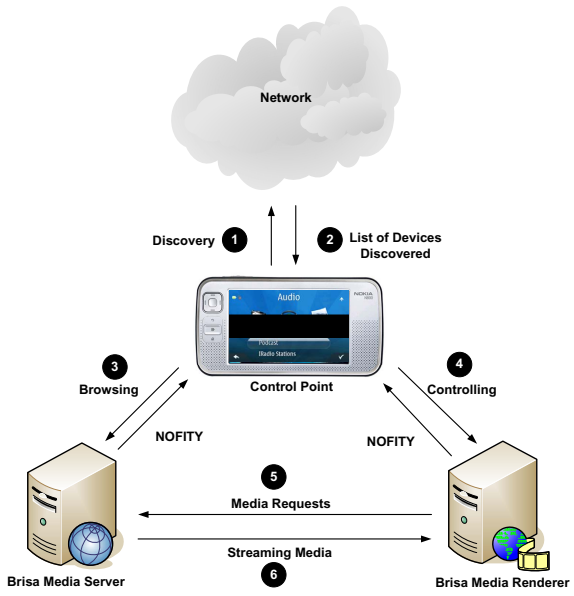

Figure 1. A/V UPnP devices communication.

Basically, there are three main entities in the UPnP A/V specification: the Media Server, Media Renderer and Control Point. The media server is responsible to share UPnP multimedia items [4] such as audio, video and image files. Similarly, the media renderer is capable of reproducing all multimedia data shared by any UPnP media server. When a control point is started up, it sends the discovery message (step 1 of the Figure 1) to search for all UPnP media servers and media renderers. Each of the available servers in the network replies to the control point request which, at this point, knows all of them. As a result, a user running the control point in his/her device can render any multimedia item shared by a media server by simply browsing a specific media server according to UPnP specification.

3. Enabling User Profile for UPnP standard

This section presents the UPnP-UP extension that enables user authentication and authorization. To achieve the UPnP-UP solution, we introduce to UPnP a new entity named UPnP User Profile Server. It is similar to the UPnP Media Server being responsible for storing user profile information, such as fullname, username, password and specific information regarding any UPnP specification such as the UPnP A/V or Home Automation. As already mentioned, this information is not available in a UPnP network. Also, the current UPnP specification just provides methods to learn about UPnP devices, not to learn about who is accessing the UPnP services. Therefore, it is not possible to provide a service based on the users preferences.

In order to solve this problem, the User Profile Server stores the users profiles into a database and exposes them through SOAP web services, just like the other UPnP services are provided. In this way, adding a new type of UPnP device does not impact in the UPnP current specification. Otherwise, it requires minor efforts to make the UPnP-UP solution easy to be implemented. Therefore, we reached one of our goals, which is making a solution strongly compatible with technologies already being used by the UPnP standard. In this case we avoid UPnP implementors learning and implementing one more technology, such as REST [5] or XML-RPC.

By exposing the user profile through web services, the User Profile Server can store the user profile by any back-end a developer prefer. For the implementation described in Section 4, an LDAP back-end was used to store the user profile running on the real domain named `http://www.upnp-up.org/`. Once the User Profile Server is defined, Figure 1 should be modified to insert the new entity in the basic UPnP A/V solution. This new configuration is shown in Figure 2.

According to Figure 2, after discovering online UPnP devices (steps 1 and 2), a control point invokes a SOAP method named *auth* (step 3), which sends the user's *username* and *password* to the UPnP-UP Server. As a consequence, it receives a user authentication ID (a random *UUID* generated by the UPnP-UP Server). This identification will be used to identify the user after a successful authentication (step 4). An important modification that should be pointed out is related to the NOTIFY message sent to the control point by the UPnP device (see Listening 1). If the UPnP device discovered by the control point supports UPnP-UP extension (UPnP Media Server and UPnP-UP Server, for instance), it must send a modified version of that *NOTIFY* message back to the control point. The new version of the *NOTIFY* message must be one as shown in Listening 2. Note that in line 6, it contains a new field

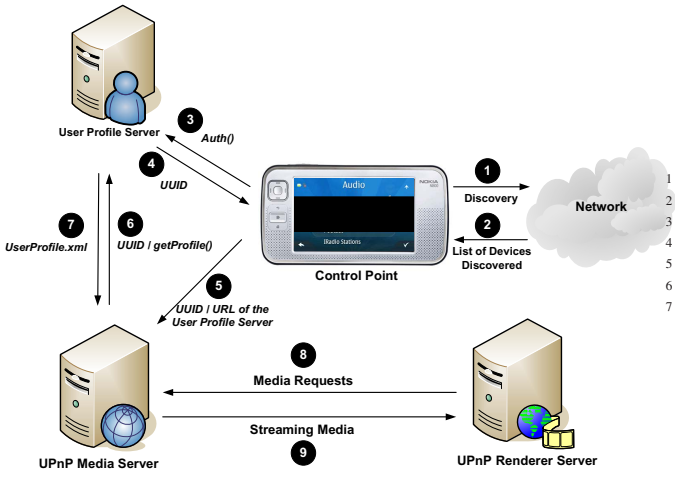

Figure 2. A/V UPnP devices communication with User Profile Server.

named UP (User Profile), which indicates whether the remote UPnP end-point supports the UPnP-UP extension.

Once the control point has been notified that the remote UPnP end-point supports the UPnP-UP extension, it is able to invoke the remote *auth* method to the UPnP-UP Server. For the schema shown in the Figure 2, it's clear to see that the control point sends a successful user authentication message to the UPnP Media Server, as well as its communication UUID and the URL of the User Profile Server where the user has logged in (step 5). Next, the UPnP Media Server requests the user profile (steps 6 and 7) to be locally cached. Since the UPnP Media Server has the user profile, it can use the user preferences to offer customized data based on actions performed by the user from his/her UPnP control point (steps 8 and 9).

```
NOTIFY * HTTP/1.1
HOST: 239.255.255.250:1900
CACHE-CONTROL: <max-age>
LOCATION: <url of device description>
NT: <search target>
UP: yes
```

Listening 2: The New UPnP NOTIFY message.

The *renewSession* SOAP method is another service exposed by the UPnP User Profile Server. The UUID sent back from the UPnP-UP Server to the control points must be valid for 300 seconds. In case of a timeout, the control point must invoke the UPnP User Profile Server's *renewSession* method to to renew their communication UUID. Moreover, this process adds a security level in the case of the user UUID is obtained by a non-authorized user.

3.1. The New Model of UPnP Devices Discovery

After the user authentication process is finished, UPnP devices should interact between each other according to the UPnP-UP extension. To achieve this process, the control point must send the user identifier in every message transmitted and the others UPnP end-points should use the user identifier to adapt their responses according to the user profile. For the example shown in Figure 2, steps 8 to 9, the current communication process for browsing into a media server must be modified. Finally, parts of a python code implemented in the media server according to the UPnP specification are shown in Listening 3. It handles a control point browse request.

```
def soap_browser(object_id, browse_flag,
                 filter, starting_index,
                 requested_count,
                 sort_criteria):
    plugin = retriave_plugin(object_id)
    items = plugin.select_items(object_id)
    return items.to_didl_xml()
```

Listening 3: The current soap browser of the media server.

With the addition of the UPnP-UP extension capabilities, the *soap_browser* function signature must be modified to accept the *user identifier*. This modification is shown in Listening 4. From this example, the idea behind the UPnP-UP extension can be used for others UPnP end-points.

```
    //previous lines omitted
    items = plugin.select_items(object_id,
                                user_id)
```

Listening 4: The UPnP soap browser using UPnP-UP extension.

3.2. The User Profile Device Description

As described in Section 2, all the UPnP devices (or end-points) exposes an XML that describes them and the services provided by the device. Listening 5 shows an example of the UPnP User Profile Device description.

In lines 16 and 27 of Listening 5, one can note the new UPnP services named UPAuthentication and UPProfile. From these information, the control points can get access to the service description from the single from a SOAP message. (see the values of the tags <*SCDPURL*> and <*controlURL*>, respectively).

```
<root xmlns="urn:schemas-upnp-org:device-1-0">
 <specVersion>
   <major>1</major>
   <minor>0</minor>
 </specVersion>
 <URLBase>base url value</URLBase>
  <device>
   <deviceType>
     urn:schemas-upnp-org:device:UPServer:1
   </deviceType>
   //some tags were omitted! See UPnP Reference
   for more detail
   <serviceList>
     <service>
       <serviceType>
           urn:schemas-upnp-org:service:UPAuthentication:1
       </serviceType>
       <serviceId>
         urn:upnp-org:serviceId:11
       </serviceId>
       <SCPDURL>up-authentication.wsdl</SCPDURL>
       <controlURL>up-authentication/control</controlURL>
       <eventURL>up-authentication/event</eventURL>
     </service>
     <service>
       <serviceType>
           urn:schemas-upnp-org:service:UPProfile:1
       </serviceType>
       <serviceId>
           urn:upnp-org:serviceId:22
       </serviceId>
       <SCPDURL>up-profiles.wsdl</SCPDURL>
       <controlURL>up-profiles/control</controlURL>
       <eventURL>up-profiles/event</eventURL>
     </service>
   </serviceList>
   <presentationURL>URL for presentation</presentationURL>
```

```
  </device>
</root>
```

Listening 5: Example of the User Profile Server Device Description.

3.3. The User Profile Description

Considering the schema shown in Figure 2, an aspect that should be pointed out is in the steps 6 and 7. When a UPnP end-point requests a user profile, the UPnP-UP Server returns a user profile in an XML format. This user profile description is an XML file format similar to Listenings 6 and 7 – similar to the basic UPnP device description XML file. There are two types of user profile descriptions:

1. **UPnP-UP Personal User Profile**: In this case, each personal user profile contains at least an identification, fullname and a username (see Listening 6); and
2. **UPnP-UP User Profile by Specification**: For each UPnP specifications defined (UPnP A/V, UPnP Lighting Controls [12] and others), a user has its associated related fields and values. For example, from Listening 7 , the tuple *upnpCategoryList* (called containers) identifies the UPnP specification through *id* and *name* attributes. For this example, the system stores user profile information for the UPnP A/V specification. As a consequence, information for this specification should be regarding user preferences for audio genre, music artist and album, the preferred video resolution, codification, or format, and so on.

The goal to split the user profile into different containers is to facilitate the process of filling out the profile form, where the user can choose which fields he/she has interests in. Another reason is to optimize the user profile access and the transmission of useful profile information to the UPnP end-point through SOAP messages. As it was discussed earlier, the strategy allows UPnP end-points to filter specific containers of the complete user profile according to the their owner's interest.

```
<root xmlns="urn:schemas-upnp-up-org:up-1-0">
 //Some tags were omitted. See UPnP Reference.
 <UserInfo>
  <userId>User Id - UPnP-UP Server</userId>
  <userName>username@upnp-up.org</userName>
  <fullName>User full name</fullName>
  <email>User email addresss</email>
  <homepage>User home page</homepage>
  {...}
 </UserInfo>
</root>
```

Listening 6: Example of the User Personal Profile Description.

```
<root xmlns="urn:schemas-upnp-up-org:up-1-0">
 //Some tags were omitted. See UPnP Reference.
<userProfile id="7569">
 <upnpCategoryList>
  <category id="1" name="AudioVideoImage">
   <genreList>
    <genre id="17" title="Rock">
     <artistList>
      <artist weight="83">Guns N Roses</artist>
       <itemList>
        <item contentType="AUDIO" weight="82">Welcome
            To The Jungle</item>
        <item contentType="PICTURE" weight="93">November
            Rain</item>
        <item contentType="VIDEO" weight="75">Don t
            Cry</item>
       </itemList>
     </artistList>
    </genre>
    //closed tags omitted.
```

Listening 7: Example of the User Audio/Video Profile Description.

4 Recommendation using the UPnP-UP

This section presents a multimedia recommendation system based on both UPnP standard and UPnP-UP extension, as described in previous sections. Recommendation systems are capable of recommending items based on user's profiles and by using specific information filtering techniques, which are applied either against to the items content or to a repository of data such as a database.

By enabling user profiles in UPnP standard, new types of systems can be designed, which should make use of the UPnP standard to search and discover devices in a network. For instance, suppose a UPnP system capable of sharing multimedia items such as music, movies and images. According to the user profile obtained by the UPnP-UP extension, described in Section 3, the system can offer songs based on the genre given by a user as the best genre he/she likes. In this way, one can develop a more sophisticated system that can rate all the songs listened by the user through the media renderer and then infer which genre he/she prefers, such as rock, jazz, etc.

With the UPnP-UP extension, this idea may be extended to other domains, such as home automation. In this case, a UPnP based application that adapts the fuzzy light level according to the user preference can be deployed in a living room, acting when the user enters in his/her private room. In this field of application, another example is to adapt the air-conditioner temperature taking into account the user preference obtained by the UPnP-UP extension.

4.1 Recommendation in BRisa Media Server

The BRisa Media Server [6] is a UPnP end-point device developed in the Embedded System and Pervasive Laboratory that implements the UPnP A/V specification. It has been written in Python using a plug-in based architecture. In order to compute the recommendation of multimedia items, the BRisa Media Server was modified to support UPnP-UP following the steps described in Section 3. After, the recommendation system was developed. It has a dynamic multi-thread plug-in loading scheme focusing on resource limited devices, mainly for the Nokia Maemo Platform [3]. Additionally, it has a database scheme to avoid loading many objects into the main memory. BRisa can be freely downloaded from `http://brisa.garage.maemo.org/`, since it is distributed as an open-source software under the MIT license.

In face of this type of scenarios and to achieve the recommendation idea, the user profile is compared to some reference characteristics. These characteristics were collected either through the information of the item – a content-based approach [1] – or by the user social environment, a collaborative filtering approach [8].

When a new user wishes to have multimedia files recommendations, he/she needs to set up his/her multimedia profile. As a result, this process consists of building his/her user profile for music, videos or audios of his/her preference. In our case-study, we collected the user profile from a public website.

As it can been seen in Figure 3, the first step is the collaborative filtering approach. For each multimedia consumed

item – a listened song, a watched video clip and so on – the user specifies a weight to the listened item considering his/her own interest. The possible values for that can be specified ranging from 0 to 10, where 0 means the worst and 10 means the best.

Following the recommendation process, the content-filtering approach can be done by selecting those items which weights higher or equal to a previously given threshold, say 5, and finding similar items with TF-IDF (Term Frequency – Inverse Document Frequency) algorithm [1]. To achieve this last approach, the multimedia file metadata was obtained through ID3 pattern.

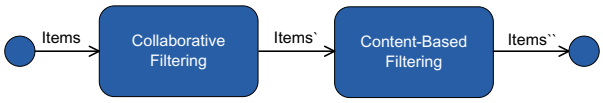

Figure 3. Recommendation flow

Another important feature of the BRisa media server implementation is the plug-in based architecture, which is shown in Figure 4.1. We developed an extensible and flexible architecture which enables third-party developers to implement plug-ins. These plug-ins can share multimedia data from a particular source. In the first step, a BRisa plug-in developer creates a plug-in file and puts it into the plug-in directory. Then, he/she creates a python class assigning a plug-in name and implementing the methods *load*, *unload*, *execute* and *browse*. In our case-study, we developed a set of plug-ins that supports browsing of multimedia contents.

Basically, three plug-ins was developed: one to gather multimedia content from the local file system, a second to retrieve YouTube videos from a personal user account and a third for retrieval of pictures from a personal FlickR user account. Each plug-in has its own data source. For example, the file system plug-in data source is the disk, and for YouTube plug-in its data source is the list of available videos from the `youtube.com` website. Regardless the plug-in selected by the BRisa Media Server, As can be seen in Figure 4.1, the *browser* method invokes the recommendation subsystem in each of them, which in its turn, recommends multimedia items to the user. The whole process adopted can be summarized as follows:

1. the BRisa Media Server receives a browsing request from the control point, also provided in the BRisa Application package;
2. the BRisa Media Server in its turn selects the proper plug-in according to which plug-in is responsible to handle the request;
3. a plug-in *i* is selected to invoke the recommendation subsystem in order to determine which multimedia content will be retrieved from its specific repository;
4. once the recommendation subsystem has access to the User Profile Server, it gets necessary information to execute the recommendation as described before, considering Figure 3;
5. the Recommendation Subsystem sends *i* the results of the recommendation back to the plug-in; and
6. the plug-ins access their respectively data source and collect the filtered items;
7. the selected items are sent back to the BRisa Media Server; and
8. finally, the set of items is formatted into a Browsing Response considering UPnP A/V specification and it is returned to the user control point.

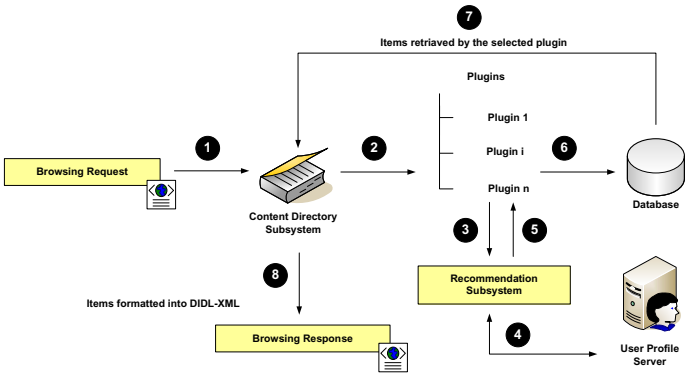

Figure 4. The Browsing Process Using the Recommendation Subsystem

5 Experiments and results

To evaluate our proposed UPnP-UP extension, we performed a series of experiments with a total of 30 users. The graph illustrated in Figure 5 shows the results of the experiments considering the recommendation quality for each user. Each of them gave a weight for his/her recommendation process ranging from bad to excellent qualities, where such qualities mean if the recommended items (music, videos or audios) matched his/her multimedia preferences.

Based on the given results, 32 % percent of the users considered that the recommendation system works good. Furthermore, 22 % of the users defined the recommendation excellent. On the other hand, just 6 % of the researched users defined the recommendation system as bad. Note that a recommendation system bases its recommendations in a set of assumptions specified by the user. As a consequence, these assumptions may not be properly specified, which may impact in the quality of the recommendation for certain users. For instance, the Collaborative Filtering has the inability to give recommendations for users with few ratings, and for this reason, these users don't have good neighborhood, because he/she is not similar to anybody else. Therefore, the Content-based Filtering is harmed, making the quality of the recommendation process to be low.

6 Current Works

As current works, a User Profile Internet Server is being implemented to be hosted in `http://www.upnp-up.`

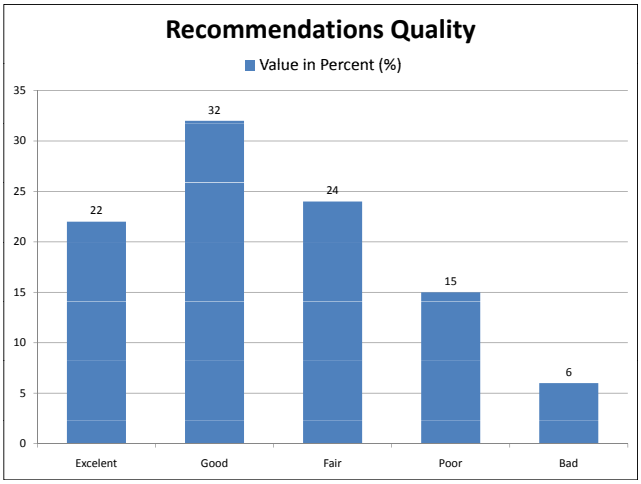

Figure 5. Results of recommendation process through UPnP-UP extension

org website. The UPnP users will be able to setup their accounts and fill out their profile. At the time of writing, the website is currently capable of sharing the user profile through SOAP web service, implementing the UPnP extension described in Section 3. Moreover, the authorization service is being developed, which will provide authorization mechanisms for the UPnP services. For this topic, we are considering the SAML (Security Assertion Markup Language) [7], that can be used to describe user permission in a system in an XML format. Similarly, an application of this service is if two control points control a single UPnP device.

Nowadays, there is no mechanism to specify which user, through his/her control point, can control or have priority to control a remote device. For example, a UPnP home automation system, using the authorization service, will be able to solve conflicts of permission problems, including when the user in a UPnP controlled house has higher permission of turning room's lights on or off considering the fact that both of them are in the same room – in this case the father should have more priority than his son.

7 Conclusions

This paper presented our efforts to enable user authentication for the UPnP standard. The UPnP-UP was proposed and explained, which extends the UPnP standard and specifies a new UPnP device named User Profile Server and a UPnP service, the UPAuthentication. As a proof of concept, the BRisa Media Server was used to implement a recommendation system based on the UPnP standard and on the UPnP-UP extension. The UPnP-UP requires minor changes to enable both user authentication and authorization in a UPnP network. Furthermore, even through UPnP-UP provides these modifications to the UPnP standard, the UPnP-UP devices still remain compatible with the current UPnP devices and services specification.

Regarding to the recommendation results obtained in the case-study defined in this work, regardless to the 6 % of users that voted as a bad items recommendation, the system performs like expected, considering the fact that the recommendation mechanisms widely used in other well-known recommendation systems were implemented. As discussed, a recommendation system bases its recommendations in a set of assumptions specified by the user, thus if these assumptions are not properly specified, this may generate an impact on the quality of the recomendation for certain users.

Finally, the UPnP-UP was possible thanks to the flexible and extensible standard offered by the UPnP Forum, which allows the addition of new devices and services through a definition of artifacts such as XML file descriptions and SOAP Web Services.

References

[1] M. Balabanovic and Y. Shoham. Fab: content-based, collaborative recommendation. *Communications of the ACM*, 40:66 – 72, 1997.

[2] B. Catania, A. Maddalena, and A. Vakali. Xml document indexes: A classification. *IEEE Internet Computing*, 2005.

[3] N. Corporation. Maemo platform, 2002. `http://maemo.org/`. Last access, August 2008.

[4] N. Corporation, I. Corporation, and M. Corporation. Upnp device architecture, 2006. `http://www.upnp.org/specs/arch/UPnP-DeviceArchitecture-v1.0-20060720.pd%f`. Last access, May 2008.

[5] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California - Information and Computer Science - UCLA, 2000.

[6] A. Guedes, D. Santos, J. do Nascimento, L. Sales, A. Perkusich, and H. Almeida. Set your multimedia application free with brisa framework: An open source upnp implementation for resource limited devices. *Consumer Communications and Networking Conference, 2008. CCNC 2008. 5th IEEE*, pages 1257–1258, 10-12 Jan. 2008.

[7] P. Harding, L. Johansson, and N. Klingenstein. Dynamic Security Assertion Markup Language: Simplifying Single Sign-On. In *IEEE Security & Privacy*, volume 6, pages 83–85, 3 2008.

[8] R. D. T. Júnior. Combining collaborative and content-based filtering to recommend research paper. Master's thesis, 2004.

[9] G. Langille, A. Presser, and M. Walker. Upnp a/v architecture, 2002. `http://www.upnp.org/specs/av/UPnP-av-AVArchitecture-v1-20020622.pdf`. Last Access, May 2008.

[10] P. Louridas. Soap and web services. *IEEE Software*, 2006.

[11] I. N. I. of Technoly. Canola project, 2002. `http://openbossa.indt.org.br/canola/`. Last access, May 2008.

[12] C. Sahm and H. J. Langels. Dimmable light device template, 2003. `http://www.upnp.org/standardizeddcps/documents/DimmableLight1.0cc.pdf`. Last Access, May 2008.