**V**rije **U**niversiteit **B**russel
Faculteit Wetenschappen - Departement Informatica
Academiejaar 2000 - 2001


# Program suggestions based on Community-Profiles

Peter Ebraert

*Proefschrift ingediend met het oog*
*op het behalen van de graad van*
*Licentiaat in de Toegepaste Informatica*

Promotor: Prof. Dr. Theo D'Hondt
Co-promotor: Prof. Dr. Patrick Steyaert

**V**rije **U**niversiteit **B**russel
Faculteit Wetenschappen - Departement Informatica
Academiejaar 2000 - 2001


# Program suggestions based on Community-Profiles

Peter Ebraert

Promotor: Prof. Dr. Theo D'Hondt
Co-promotor: Prof. Dr. Patrick Steyaert

**Abstract**

Deze thesis onderzoekt de mogelijkheden van **personalisatie in het broadcasting milieu**. Drie grote stappen moeten genomen worden om een programma te maken dat in staat is om een persoonlijk TV programma op te stellen: Data Gathering, Community Detection en Building en de Program Suggestion zelf.

Er zijn drie mogelijke technieken om TV progamma's aan te bevelen. Zo zijn er de individuele methode, de collaboratieve methode en de community gebaseerde methode. In deze thesis onderzochten we de verschillende mogelijkheden van de hierboven vermelde technieken. Als ondersteuning van de discussie werden enkele experimenten uitgevoerd.

Het is duidelijk dat deze innovatieve technologien zowel voordelen als nadelen zullen hebben, afhankelijk vanuit wiens oogpunt er naar gekeken wordt. De gevolgen van de personalisatie op de betrokken partijen zullen bestudeerd en besproken worden.

Ten slotte wordt er een experimenteel onderzoek besproken om de resultaten van de verschillende methoden te staven.

**Abstract**

This thesis investigates **personalisation in relation to television**. Three main steps have to be taken to make a TV program suggesting device: Data Gathering, Community Detection and Building, and the Program Suggesting itself.

There are three possible techniques to suggest different TV programs to the users: the individual method, the collaborative method and the community based method. In this thesis we have investigated the possibilities of the above-mentioned approaches. A number of experiments were performed to support the discussion.

It is clear that this innovative technology will have both benefits and drawbacks depending from which viewpoint it is looked at. The consequences it has on the concerned parties will be made clear.

Finally an inquiry was organized in order to validate the results of the different approaches.

# Acknowledgement

First of all, I would like to thank **Prof. Dr. Patrick Steyaert**, **Wilfried Verachtert** and **Fillip Vertommen** for allowing me to do my thesis based on experiences at MediaGeniX. I also want to thank them for the support they gave me when needed, for their punctual proof reading of several drafts from my thesis, and especially for the close cooperation I experienced during the development of my thesis.

For her great support during the hardest days, I would like to thank my girlfriend **Sabine**, without whom's care and special attention the work would have been a lot harder; and **my parents**, who granted me the opportunity to study and supported me during my studies.

Finally, I want to thank **Kris** and all my friends who supported me during this difficult period. A special thanks goes out to **Michael Schindler** and **Dirk Deridder** who helped me getting rid of grammatical and spelling mistakes and provided me with useful tips on several practical issues.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Have you ever been in the following situation? You are looking for information on the internet, but you always get information you are not interested in. I suppose that most of the people who browse the net from time to time have already experienced that problem. The logical explanation for this is that there is an excess of information on the net.

Ever since 1998 the end user of computer systems has become increasingly more important. Knowing that the majority of the end users is no computer-scientist, it is obvious that each system must be *usable* (= user friendly). Therefore the new User-centered design[1] is used more and more in programming. These two facts show the need for *personalisation* in software systems [23].

The same is happening in the TV industry. More and more television stations start up, and the offer of different programs increases each day. Today, in Belgium (a country of only 10 million inhabitants), 30 different channels can be watched. In the US the situation is even worse, with over 100 channels to choose from! In addition television will change entirely in the near future [19]. One possibility is that television stations with only one target-community will be raised. Those stations will *only* broadcast programs for a certain target group. From the moment broadband is available for everybody, the TV-stations will no longer broadcast *their* schedule, but the user will be able to choose his own schedule through *television on demand*. Turning on his television, the user will be shown a list of possible programs. He will be able to look at each one of them. This image of the near future shows how big the need for personalisation is. Can you imagine turning on the television and having to choose from about 1000 titles?

The alternative lies in an application that knows your interests and which shows only the list of TV programs which correspond to those interests. Such a system can be developed using a *program suggesting* method. There are three different methods that can be used to suggest TV programs. The first method, *the individual method* only uses the personal profile of a user.

Matching the Program characteristics with the user profile, it will know whether the program has to be suggested or not.

The second method, *the collaborative method* is based on the assumption that the users with the same interests will have more or less the same opinion on a certain program. It will use that assumption to know whether to suggest a program or not.

Finally there is *the community-based method*. It makes use of community-detection. Groups of users with the same interests will be detected among the entire user-set. Knowing that a community contains all people with the same interests, a community profile can be composed. This profile will be used to recommend unknown programs with more certainty of succeeding.

## 1.1 Problem setting

The surplus of information these days and the need of usability of computer systems lead to the need for **personalisation** [4]. Through personalisation, we can make a system which only displays the relevant information to the end user [6]. Also the way the information is displayed will be personalised.

Another example can be found in the banking industry. The key challenge faced by banks today is how to offer the service provided by the one-to-one relationship costumers had with their bank manager, but at low delivery costs. Bank services and products have largely become commodities. So if they want to be competitive, they should offer something extra. There are three methods of achieving this: further defining their brand, following a lower cost cost delivery strategy or looking to offer more personalised service to the costumer. [3] An extra difficulty is the large amount of clients a bank has. Personalisation for all the costumers has to be ensured.

Offering the user the information he was looking for and allowing him to display that information his way, will add value to the product and will create a kind of loyalty from the costumer. [7] But we also have to ensure that the user trusts the system. He has to be convinced that the price he has to pay (= giving up control) is worth the advantages of personalisation. [9] The laws made on *the users privacy* are inadequate for guaranteeing *the right to privacy* of the users. [14] Therefore it has to be guaranteed to the user that the obtained information will not be used for other purposes than providing personalisation. We should also convince them that personalisation indeed gives a better service. For that, we should measure the happiness of the public and adapt the system if necessary. [8]

Experience teaches us that creative users will more easily try out a new program than conservative users. Therefore we also have to ensure that creativity is supported in the system. This can be done by making the system as flexible as possible. [10] One way of doing this, is making our PC *intelligent*. We would have to provide it with commonsensible knowledge so that it can understand us. But in artificial intelligence, many attempts have already been made to do this, without success. A second way is to let the user make personalised tools he wishes to use. [7], [15]. We should also allow 2 kinds of personalisations: what is displayed and how it is displayed. [13] This leads to the following requisite: we should use standard representation for the storage (such as XML) of information. That information can afterwards be displayed in the way the user chooses. [16], [17]

This thesis will focus on personalisation in the television world. We will compare three different program suggestion methods. The following questions will be answered. How can we suggest certain programs? Which technologies are used? What are the consequences of this new system for all the involved parties?

## 1.2 Goal

The main goal will be to compare and evaluate the different program suggestion methods in the world of broadcasting. Not only the practicality, but also more profound aspects; like ethical consequences, architecture, functionality and performance, will be discussed.

For testing those methods, a personalised television program scheduler must be set up. There are some possible solutions for making such a system. We will have to find a solution that copes with the problems cited in the previous section. We must be aware that the system is scalable, so that the system can cope with various numbers of users. The privacy of the users should be guaranteed so that they completely trust the system. The standard representation language (XML) will be used for storing and transmitting information. The requested (and only relevant) information should be displayed on screen (in a way the user likes it).

In addition the user has to be convinced that the service he is being given, is worth giving up control. Therefore we have to check the results and adapt the system if necessary. So a continuous follow-up of the system is required. Therefore we need the continuous feedback of the users of the system. We want that at least 90% of the users is satisfied with their program recommendations.

Finally we want to obtain the users opinion on the different program suggesting methods. Therefore an inquiry will be organised.

## 1.3 Setting up the system

Building a system that provides a schedule on personal and community preferences can be divided into three big steps [18].

The first step is providing the information. For that, a web page had to be built which was reachable for most people. The page consists of some views on the different programs broadcasted that week. If the user wants to know more about some program, he has to click that title. After that, he receives more information on that broadcast [22]. When the user asks some information about a certain program, we can assume he is interested in that show. We store this fact in his data-sheet. After a while, the data-sheet of a user will represent the users preferences. With those preferences, a *user model* can be built. The collecting and the saving of the information is called the *data gathering*. The way we obtain this information is variable and will certainly change over time [12]. An implemented example of this step is discussed in chapter 3.

The second step is how we can divide the users in different groups with their specific characteristics. This part has multiple possible solutions: *data-mining*, *rule-engine*, *classic programming* and *Machine Learning*. I used a rule-engine in my implementation, named JESS. This shell has a big advantage on the other rule-engines; it is free and open source. This is called *community-detection and -building*. A comparision between the different solutions will be included. An imlemented example of this step is discussed in chapter 4.

After having grouped the users in a number of communities (all with their specific interests), we have to provide a schedule based on the preferences of the community and/or the personal preferences of the user. Three different methods can be used to do this. We will compare the different methods in chapters 7, 8 and 9. This last part is called *Personalisation*. Personalisations include both *layout* and *content*. So the way the information is displayed on screen will also differ according to which community the user belongs to.

## 1.4 How to read this thesis

After a general introduction on the different technologies used, we will take a closer look on the different steps we have to take to make a TV program suggestion system based on communityProfiles. Those are the gathering of the data to compose userProfiles, the detection and building of communities and the way the personalizations can be realized. We will not try to give an overview on the different methods of how this can be accomplished. But we will discuss one practical implemented example. This should give the reader an image of how the different steps **can** be solved.

At the end of the first part, the possible applications of television related personalisation will be described. Also the consequences they will have for all the concerned parties will be discussed: as there are the broadcasters, the viewers and the publicity companies.

In the second part, we will discuss three different methods which can be used to provide the users with a personalised TV schedule. The advantages and drawbacks of each method concerning performance, architecture, ethics etc. will become clear in this part. At the end of the second part, we validate our results through an inquiry. We will discuss how the users react on the different suggestion methods. Throughout this thesis, we will assume the number of keywords that characterize a TV program to be variable but the same for all the TV programs.

# Part I

# Preliminaries

# Chapter 2

# Study of environment

In this chapter an overview will be given of the programming languages that were used during the study: HTML, XML, XSL, DOM, JSP and JESS. Also the cooperation between those languages should become clear after reading this chapter. I will not try to explain their syntax, but rather focus on the meaning and purpose of those different languages. These standards and languages can be divided into three groups. The first group, containing HTML and XML is the data-describing group. The Standard Generalized Markup Language (SGML) is the basis of these two essential Internet standards[1]. The second group is used for presenting the information. In this category we can classify DOM, XSL and JSP. Finally there is JESS. JESS is used as a rule-engine for deriving new knowledge from existing knowledge [2].

## 2.1 Hyper Text Markup Language, HTML

If you are familiar with HTML, you can easily skip this section, because I will only give a short introduction to it here. For more information I refer to [47].

HTML is used for writing down the contents of web pages. A web-browser (like Internet explorer or Netscape) will know how to *translate* such a web page into readable text with a nice layout. For doing this, the web page contains the content that has to be displayed, but also the way it has to be displayed. Tags do this as you can see in the following example:

```
<p>This is a Paragraph.</p>
```

Interpreting this, the web-browser knows he has to display this sentence as a paragraph[29]. Netscape, Microsoft or another company does not make up the standards of these tags. They are put together by the W3C (World

[1]For more information see [28]

[2]Tutorials on these languages can be found on [2]

Wide Web Consortium)[30]. That way, HTML has now grown out to be "the" web-page-standard. Other standards include CSS[31] and XML. The consequence is that the web can be surfed by everybody (having different web-browsers and even different Operating Systems). That is why the use op HTML enhances *portability* of information.

## 2.2 eXtensible Markup Language, XML

XML is a markup language much like HTML[32]. But whereas HTML is used for making web pages, XML was designed to describe data. Since XML is a markup language, it should not be surprising that it also makes use of tags. The big difference between XML and HTML is that there is no standard for the tags in XML. Here your own tags must be defined. Note that capitals are used for the tags as a convention.

```
<NOTE>
  <TO>Sabine</TO>
  <FROM>Peter</FROM>
  <HEADING>Reminder</HEADING>
  <BODY>Don't forget me this weekend!</BODY>
</NOTE>
```

This example intuitively describes a note from Peter to Sabine. A clear hierarchy of tags can be seen. XML uses a DTD (Document Type Definition) to describe the content. It gives a description of its format. Therefore an XML-document with a DTD is designed to be *self-describing*.

XML can best be described as a cross-platform, software and hardware independent tool for transmitting information [2].

I strongly believe that XML will be as important to the future of the Web, as HTML has been to the foundation of the Web, and that XML will be the most common tool for all data manipulation and data transmission [33]. But just like HTML, XML *does* nothing on its own. It only describes some data. That is why we need something to display that data, like XSL or through the DOM.

## 2.3 XSL and DOM

Here I will give a short introduction to XSL and DOM. I will also give a comparison between the two based on my experiences.

### 2.3.1 eXtensible Style sheet Language, XSL

Because HTML uses predefined tags, the meanings of these tags are well understood: The `<p>` element defines a paragraph and the `<h1>` element

defines a heading, for instance. The browser knows exactly how to display these elements. Adding display style characteristics to HTML elements with CSS is a simple process. Telling the browser to display each element using a special font or color is easy to do and easy for a browser to understand.

Because XML does not use predefined tags (we can use any tag we want), the meanings of these tags are *not understood*: `<table>` could mean an HTML table or maybe a piece of furniture. Because of the nature of XML, the browser does not know how to display an XML document. In order to display XML documents, it is necessary to have a mechanism to describe *how* the document should be displayed. XSL is the preferred style sheet language of XML, and XSL is far more sophisticated than the CSS used by HTML.

The following XSL code will, when called on a XML file, output an HTML-file with a table in it containing all the Notes with the corresponding fields (FROM, TO, BODY). For more information see [34].

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
  <html>
  <body>
    <table border="2" bgcolor="yellow">
      <tr>
        <th>From</th>
        <th>To</th>
        <th>Text</th>
      </tr>
      <xsl:for-each select="NOTE">
      <tr>
        <td><xsl:value-of select="FROM"/></td>
        <td><xsl:value-of select="TO"/></td>
        <td><xsl:value-of select="BODY"/></td>
      </tr>
      </xsl:for-each>
    </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

*All together, XSL is used for transforming XML to HTML.*

### 2.3.2 The XML Document Object Model, DOM

A program called an XML parser can be used to load an XML document into the memory of your computer. When the document is loaded, its information can be retrieved and manipulated by accessing the DOM. The DOM represents a tree view of the XML document. The *documentElement* is the top-level of the tree. This element has one or many childNodes that represent the branches of the tree.

A *Node Interface Model* is used to access the individual elements in the node tree. As an example, the childNodes property of the documentElement can be accessed with a for/each construct enumerating each individual node.

The *Microsoft XML parser* supports all the necessary functions to traverse the node tree, access the nodes and their attribute values, insert and delete nodes, and convert the node tree back to XML.

*How does it work?*
You start a script in your HTML page. In that script you use your chosen programming language to load the XML code into an XML parser. After that you can start accessing the required data through the outputted DOM[35].

### 2.3.3 Comparing XSL and DOM

As you could read in the two previous sections, there are two main ways to transform XML data to a readable HTML. I should say that XSL happens on surface level, while the DOM goes to the core. XSL is a language that is used to directly access the right data out of an XML file. You use the names of the fields to get the requested data out of the XML file. You need to *apply* an XSL file on an XML-file to get the desired result.

Through the DOM you can also access XML data. The big difference is that this goes by means of a tree. You can access the data fields of the database through the child-parent relation in that tree. This results in a more complicated, hard coded and so more powerful way of getting the desired results.

Personally I prefer working with the DOM, because it is more powerful. Especially when you are working with JSP. In that case you are working with Java and you can manipulate and easily access the needed data fields from your XML database. For more information on XML and DOM, please refer to [36].

## 2.4 Java Server Pages, JSP

Java Server Pages technology allows Web developers and designers to rapidly develop and easily maintain, information-rich, dynamic Web pages. As part

of the Java family, JSP technology enables rapid development of Web-based applications that are platform independent. Java Server Pages technology separates the user interface from content generation enabling designers to change the overall page layout without altering the underlying dynamic content.

Java Server Pages technology uses XML-like tags and scriptlets written in the Java programming language to encapsulate the logic that generates the content for the page. Additionally, the application logic can reside in server-based resources (such as JavaBeans component architecture) that the page accesses with these tags and scriptlets. All formatting (HTML or XML) tags are passed directly back to the response page. By separating the page logic from its design and display, a reusable component-based design is supported. JSP technology makes it faster and easier than ever to build Web-based applications.

Java Server Pages technology is an extension of the Java Servlet technology. Servlets are platform-independent, 100% pure Java server-side modules that fit seamlessly into a Web server framework. They can be used to extend the capabilities of a Web server with minimal overhead, maintenance, and support. Unlike other scripting languages, servlets involve no platform-specific consideration or modifications; they are Java application components that are downloaded, on demand, to the part of the system that needs them. Together, JSP technology and servlets provide an attractive alternative to other types of dynamic Web scripting/programming that offers platform independence, enhanced performance, separation of logic from display, ease of administration, extensibility into the enterprise and most importantly, ease of use. For a general introduction on JSP please refer to [37].

## 2.5 Java Expert System Shell, JESS

Jess is a rule engine and scripting environment written entirely in Sun's Java language. Jess was originally inspired by the CLIPS expert system shell, but has grown into a complete, distinct Java-influenced environment of its own. Using Jess, you can build Java applets and applications that have the capacity to *reason* using knowledge you supply in the form of declarative rules. Jess is surprisingly fast, and for some problems is faster than CLIPS itself (using a good JIT compiler, of course.)

We will make use of JESS in the second part of the system: the community detection. We will use it to detect **clusters** within the user set and so classify the users into communities.

Like JSP, JESS has a close cooperation with Java. That is why JESS was the chosen rule engine. In our system we have to classify Java-objects (users) using JESS. Therefore we have to import those objects into the shell.

After that, we have to get the resulting classification back to Java. So if it were not possible to call methods on Java-objects out of JESS, it would not have worked out. Fortunately it is possible to call methods on Java-objects out of JESS.

In JESS all the knowledge is represented through assertions. The rules are composed of 2 parts: a left part and a right part. The Left part of a rule consists of a logical expression of assertions. And the right part consists of actions, which have to be undertaken. More information on JESS can be found on [27]. For the API of the JESS-library refer to [25].

```
(defrule do-change-baby
(Baby-is-wet)
=>
(Change-Baby))
```

The rule automatically fires when the left part (= the IF part) of a rule is an assertion which is found in the knowledge base of JESS. So here: if the baby is wet, we have to change his diaper.

Two issues should be kept in mind with this example: the way we have to represent the information in JESS is crucial for letting JESS do its work. Secondly the rules we use must be designed so that they cannot get into deadlocks.

The problem with JESS is that the execution is **non deterministic**. That means we never know which rule will be executed at what time. This brings along an unpredictable behavior and makes it rather difficult to debug. But a good rule-architecture and a good knowledge representation should help you getting started.

## 2.6 Conclusion

In this chapter we introduced all the needed technologies for making a personalization application for suggesting TV programs to the user. Now the global picture of HTML, XML, XSL, DOM and JSP should be understood. You should know how they work together to form big *portable* systems.

- It turned out that HTML was designed to display data, and to focus on how data looks while XML was designed to describe data, and to focus on what data is.
- We discussed the difference between XSL and DOM. You should know that the DOM is a more powerful tool, which will be indispensable in big systems.
- JSP provides a way to embed Java code in your HTML pages. It allows powerful and easy manipulations of XML-data.

- JESS is a rule-engine, which is freeware. It is also a plug-in for Java.

As we could see, Java is the key of the entire system. Therefore we will certainly need a Java Virtual Machine on the system where we are putting the program suggesting code on. Also the necessary Java packages should be imported to provide the necessary functionality. For more information on java see [26]. For the API of the JAVA-libraries check out [24].

# Chapter 3

# Data Gathering

Data gathering is the collecting of information on a certain topic. This can be done in two different ways: through observation and through questioning.

In this chapter we will discuss in which way the necessary information can be obtained to build suitable user models. Note that this thesis does not focus on the data gathering. Nevertheless data is needed to work with, so it is obvious that we must mention how the gathering was done. The data is necessary for composing **userProfiles**. A userProfile should describe the preferences that user has. It should tell us what kind of programs the user likes and which kind of programs he dislikes.

## 3.1 The web site

The person who wants to know something about the TV schedule for this week should connect to the web site. First he is asked his login name and password. After that is done successfully, a main menu is presented. This menu presents three choices for looking up a certain program: *part of title*, *a query on channel or/and time* or *the list of actors performing on TV that week*.

The site contains the entire schedule of TV programs for one week. The user has the possibility to look for the program he wants to know more about. When the required program is found, the user can retrieve some information about it by clicking on the program title. In that case the requested information will be displayed on screen.

Information about the actors can also be displayed. When a user clicks on an actor's name, an overview containing the personal sheet of that actor will appear on screen.

Behind the scenes, the system keeps track of all the clicks the user made. When the user clicks on a certain program, a hit will be given on all the *keywords* the program is asserted with. It is clear that in this example an observational method is used to gather information.

```
The Simpsons:  series, cartoon, youth
```

So, when somebody clicks on 'The Simpsons', a hit will be given for series, cartoon and youth.

When somebody watches the Simpsons, it can have several different reasons: that person likes *cartoons*, *series*, *youth programs*, or a combination of those. So if we see that the same user also liked another program with the keywords *cartoon* and *youth*, we can suppose that that person likes programs with the keywords *youth* and *cartoon*. That way, we can recommend him a program with those characteristics.

## 3.2 The Databases

We will need three databases for this system:

**The Program DB** :

This XML database contains all the data on the transmissions that are being broadcast during one week: the channel, the starting time, the ending time, the title, the list of genres and the list of actors are the main data fields of this data base.

**The Actors DB** :

As the name suggests, this XML database houses the data on all the actors. It is not necessary that all the actors from the program database also appear in this database. Normally this database is supposed to be a lot bigger than the previous database. It is also supposed to contain more general information (date of birth, city of birth, programs he or she has performed in, ...)

**The User DB** :

In this Database we will store the information on the users. Such a data-sheet contains some personal information (name, login, password), but also the *UserProfile*. A UserProfile is a Vector of pairs. The first element of the pair is the name of the keyword. The second element is the number of hits the user gave to that keyword. This is a relational database stored as a Java object.

## 3.3 The Server

For making this website available for use, we need to start up a server. Because we use JSP files in our application, that server must be capable of coping with them. As we have seen in chapter 2, JSP files must be compiled on server side to a servlet which outputs a HTML formatted file. That

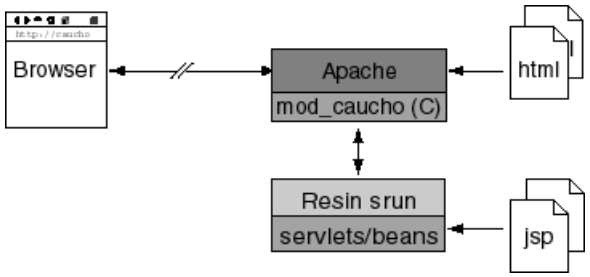


Figure 3.1: The Server Structure

HTML is then sent to the client's browser, which interprets the HTML and transforms it to a readable text with a nice layout.

So, for using JSP-files we must set up a server who can cope with compiling JSP as you can see in figure 3.1. The server must be able to run servlets and send the resulting HTML files to the clients. In this example, we work with an *Apache* server. It is a server that runs on almost every existing operating system[39].

*Resin* provides a fast servlet runner for Apache, allowing Apache to run servlets and JSP files[40].

## 3.4 Implementation

Because we need a close cooperation between our site and the User database in Java, it is almost always advisable to use JSP to implement the web pages. JSP gives the possibility to add some Java-calls as a reaction on an event. If we look at the pseudo-code below, it is obvious that some actions have to be undertaken when a certain event occurs on the web-site. Therefore in almost 90% of the pages we implemented, we used JSP.

```
Checking the login names and passwords.
When clicking a title, the keywords have to get a hit.
```

In JSP we can easily include Java objects and call the necessary methods on these objects. The following command includes all the classes of the package personalization in the JSP file.

```
<%@ page import="personalisation.*" %>
```

After this command is written in a JSP page, all the methods of the classes in those packages can be called to the corresponding objects.

## 3.5 Future

When we embed the system in a set-top box which is connected to the TV, we will not need the website anymore. The set-top box will then know exactly when the user was watching a certain TV program. The userProfile will then be stored in the set top box. The system will also have a function to rate a program in order that we know which kinds of programs the user likes. Whenever needed, the users profile will be adapted. Then a combination of both observational data gathering and data gathering through questioning can be used.

The XML TV schedule should be provided to the set top box of the client. So that the keywords, which are linked with the program are known to the data gathering program.

We can already see here that a Java Virtual Machine will be needed on the set top box for dealing with the programming code.

## 3.6 Conclusion

In this chapter an overview was given on the basic aspects of the Data Gathering. We discussed the functionalities of the website and what happened behind the screens of the web application.

A second point we mentioned was the use of the different databases. The different types of databases were discussed and also where they were connected to the application.

After that the server structure was mentioned. How *Resin* worked together with *Apache* to permit the use of JSP files. The use of JSP was necessary to link HTML with java, for permitting further processing with the collected data. We ended up concluding that JSP was used in almost 90% of our web application pages.

Finally, a short view on the future was given, on how the data-gathering environment would look like after embedding the system in a set top box.

# Chapter 4

# Community Building

In this chapter we will discuss how we can classify the users in groups to form a *community-set*. A *community-set* is a set of *communities*. A *community* is a group of persons who share the same interests or have the same characteristics. The aim of *community building* is to obtain a community-set that classifies the highest percentage of users[41]. An example of a community can be found at:

`http://ytv.yahoo.com/fc/ytv/friends`

This site is made especially for the fans of the popular TV-show 'Friends'. There the fans can obtain all the information they are looking for. But somebody who is not interested in 'Friends' will not find anything interesting on this site. It is clear that it *only* targets the Friends-community[42].

In our system we will look for people who like the same kinds of programs and then group them into communities. The resulting set of communities will have to classify more then 90% of the users.

## 4.1 Web community versus real life community

How does a Web community differ from one in the real world? In terms of their social dynamics, physical and virtual communities are much the same. Both involve developing a web of relationships among people who have something meaningful in common, such as a beloved hobby, a life-altering illness, a political cause, a religious conviction, a professional relationship, or even simply a neighborhood or town. So in one sense, a Web community is simply a community that happens to exist online, rather than in the physical world.

But being online offers special opportunities and challenges that give Web communities a unique flavor. The Net erases boundaries created by time and distance, and makes it dramatically easier for people to maintain connections, deepen relationships, and meet like-minded souls they would

otherwise never have met. It also offers a strange and compelling combination of anonymity and intimacy that brings out the best or worst in people's behavior. It is nearly impossible to impose lasting consequences on troublemakers, and yet relatively easy to track an individual's behavior and purchase patterns which makes Web communities notoriously difficult to manage. To complicate matters further, the legal issues involving privacy, liability and intellectual property on the Web are just beginning to be addressed, and will evolve rapidly over the next few years. More info is available in [41].

## 4.2 Possible solutions

The three main goals of the community detection phase can be listed. First we want at least 90% of the users to be classified in at least one community. Secondly, we want the number of resulting communities to be as low as possible. At last, we want the less possible overlapping between the different communities.

To achieve those goals, we have to compare the different interests of the users. We also have to keep in mind we do not want too much overlapping between different communities. This overlap should mean that a user is classified in multiple communities. A user who fits in two communities, it will be a lot harder to recommend programs. That is the reason why we want the resulting community-set to be disjoint.

There are multiple ways of achieving this. Here we chose JESS as a tool, because MediaGeniX wanted to know the possibilities of this rule engine. Apart from this, it is free of charge and matches our requirements.

Other possible ways include Data Mining, Machine Learning and classic programming techniques. All of them have their drawbacks and advantages. Further investigation should give a clear image on which technique is the appropriate one. Matthias Goderis is investigating more on Community detection and Community building.

## 4.3 The detection process

### 4.3.1 From JAVA to JESS

Only one way of community detection will be discussed here: the method where the Jess rule-engine is used.

A User Object holds the personal information on that user (name, login, password,...) and the profile of that user. The most important method this Class has, is the `GetMostPopular(N)` method. That method returns a vector with the N most popular keywords for that user. Using that method,

we know which interests the user has, and so JESS can figure out where to classify him.

Now we have to change this information from JAVA-format into a JESS-format. This will be explained in the next section. We start parsing the Program database. Whenever we encounter a new keyword, we make a new COMCAT association in JESS. A COMCAT association represents the association between a community and a Category[1].

When that is done, we start adding the Users in Jess through UCAT's[2].

When everything is put in, we can start the RETE-algorithm[3]. That command is given from a JAVA-main-file. The rules have three functions:

- To categorize users in communities: Creating UCOM's (= associations between a User and a Community).
- To prune away empty communities: Whenever a community does not contain enough users, it should be filtered away. Otherwise the algorithm will try to merge that community with the others, without obtaining any result at all.
- To merge big communities in order to obtain more specialized ones: A community of 1000 people which is interested in 5 different keywords, is a lot more interesting then a community of 2000 people which is interested in only series. This is because the amount of information we have on the former is a lot larger then the information we know on the latter.

When no more rules can be fired, the RETE-algorithm stops and the control is returned to the JAVA-file.

### 4.3.2 From JESS to JAVA

To get the results from JESS to JAVA, we make a ToJava rule. We are interested in the assertions of this type:

```
(COM (CID ?cid) (NR ?nr))
```

Associations of this type represent the communities. So what we have to do is to know which users and which categories are associated with that community. In the Right part of the rule we call a Java-method to make a Community Object. Then we look for all the associations between that community and the Users. For each association we find, we call the `addUser()` method on the Community Object. After that we do the same for the Categories.

[1] An Object associated with a unique keyword
[2] Association between a User and a Category
[3] The algorithm that fires the rules

The result is a Java Object that represents a Community with a list of users and a list of representative keywords.

## 4.4 Conclusion

In this chapter, we discussed what communities are and how they can be detected in a set of users.

There are four main ways to achieve this goal: through machine learning, data mining, classic programming and the use of a rule engine. In this chapter, we studied and discussed the latter.

There are 5 steps in this process:

- Getting the information of the users in the JAVA UserObject.
- Exporting the user information from JAVA to JESS
- Letting JESS fire all the rules
- Importing the community information from JESS to JAVA
- Using the information to offer advanced personalization

One possible application of this technology is providing a personal and community-based TV schedule for a user. A more commercial application is the sending of publicity to only one community. In this way a lot of money can be saved. Probably in the future, a combination of both will prove to be successful.

# Chapter 5

# Applications based on interviews

This chapter is based on interviews held with: Michael Harries from MediaGeniX, Peter Suetens of EVRT, Tom Thijs and Janick Beckers of VTM.

There is undoubtedly a huge potential in the Personal TV concept, whether it be via a *push* model and local storage, or the *pull* of material on demand. However, with either mechanism, the quest is still on for viable business models. Early adopters have been enthusiastic, but are not necessarily a reliable indicator of mass consumption patterns.

The enabling technology is now maturing – the cost per unit of hard disk storage is dropping fast and will continue to do so. This will allow storage of hundreds of hours of local material within five years. Similarly, broadband is predicted to become a reality in most territories within the next two years.

Most players in the market now believe that there is no single 'killer application', and that a whole range of services will be needed to drive viewers to take up the technology.

However, all business models that exploit the Personal TV technologies rely heavily on *metadata*. Currently, companies such as ReplayTV and Yes Television address this by compiling schedule data themselves. However, this is not sustainable once content from multiple service providers becomes available to a device, and industry standards are required to ensure consistency. For example, when a viewer clicks on a programmed trailer, there will be the option to record all episodes of that series. How will the system know what programs belong to this series?

This emphasizes the importance of high quality metadata and how to protect content rights. Those are the two main challenges facing those wanting to implement Personal TV.

A bigger problem arises when some scenarios require *a greater than program level of detail*. For example key events such as goals from a football match should be marked. The provider will be the only entity that can offer

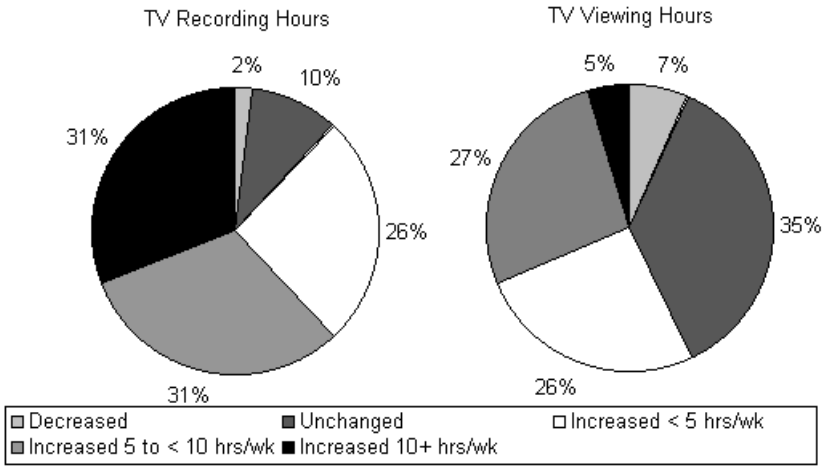


Figure 5.1: The recording and TV watching differences

such information.

Broadcasters face a challenge, as many have built an expertise in scheduling content in 'channels' that have been marketed as brands. Such channel-based brands will have little value in the future, as the viewers will be able to schedule content themselves. This implies that program brands will become more important, and that broadcasters must find other means of capitalizing on the trust they have built up over the years.

There are also social and cultural issues in this domain. The end of scheduling implies the end of the shared viewing experience, although mixed genre channels will probably survive for many years. In addition, many of the proposed business models assume that viewers will provide sufficient personal information to allow targeted advertising. This assumption has already been challenged in early applications in the US, where *privacy* has become a key concern.

## 5.1 The Impact of Personal TV on the Users

The impact on the users will be mainly positive. The users will obtain more possibilities for watching TV. Especially when classic broadcasting will continue to exist. Personal TV will then mean an extension of the already existing possibilities. The needs of the viewer are stated below:

- **Convenience**: I know what I want and I want it now, I don't want to navigate through multiple screens to get it, I want everything in one package

- **Control**: I want to be able to pause, fast forward, rewind whenever I choose
- **Choice**: I want to have a wide selection to choose from, but not so much that I am overwhelmed

Personal TV can easily supply all these needs. After the release of the application in the US an inquiry was held to measure the changes personal TV made on the users' behavior. In this inquiry was found that the overall TV viewing had increased, the recording had increased greatly (from 4 to 11hrs) and 67% of users had watched programs they would not normally have watched. The results of the inquiry are stated in figure 5.1.

Privacy of the individual has become a big issue in the US, mainly in the light of concerns over the Internet, but this has spilled over into the TV industry. Viewers therefore have the choice as to how much information they reveal about themselves (presumably this impacts on the effectiveness of the personalization of content?)

## 5.2 The Impact of Personal TV on Public Broadcasters

A common reaction among broadcasters when they hear of some of the PVR's[1] capabilities is one of **fear**, as they believe that their business will be undermined.

With a linear transmission, the broadcasters could easily create a loyal public. Now the users will not consume the television schedule linear anymore, so the channels need another way to bring loyalty. They have to make sure they don't become a simple content-provider. **Marketing** becomes more and more important for promoting their channel. The channel must be seen as a **brand**. It must target a specific community of users, creating a certain image of quality.

Offering a good selection of TV programs can do this. The broadcasters must do the editing of and the providing of subtitles to the TV programs. There are technologies used on DVDs, which enforce viewing of distributor logos at the start of a film. We see that scheduling becomes less important, while the content gains importance.

There are also some benefits included for the broadcasters. Those benefits include:

- **Better use of airtime**: More programs can and will be watched.
- **Established programs are strengthened**: Programs with high appreciation will be watched more.

[1]= Personal Video Recorder

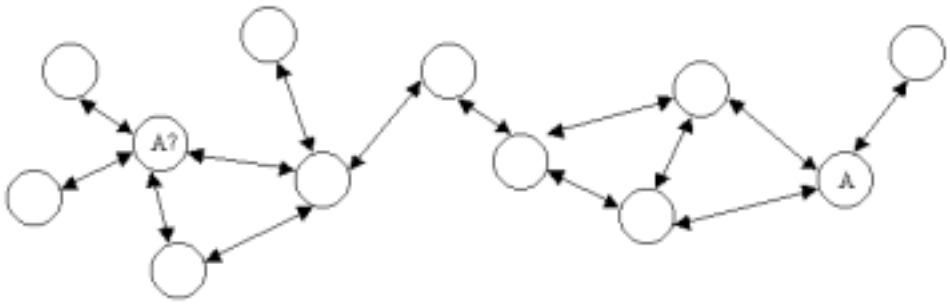


Figure 5.2: A distributed network.

But together with these advantages we see that new programs will be more difficult to introduce. How do you promote content if the viewer can skip through the trailers? The only thing that can be done, is making publicity on different media (radio, internet, PDA, magazine, ...). Trailers will be longer and only provided on demand. And whatever happened to serendipity - where you find yourself enjoying a program you wouldn't intend to watch? This will completely disappear.

Another aspect is publicity. Here the broadcasters see the creation of **events** as a solution. If commercial targeting is possible, the gain is very big. In that case the broadcasters would be able to send commercials to a particular group of people. This would cause less publicity-waste. Also the creativity of the advertising agency will be the key of success.

## 5.3 Personalization in other domains

Personalization is so important that it is almost applicable to every domain. From search-engines to news bulletins, from e-commerce to operating systems. A huge help in the personalization process is the detection of communities. If we are able to detect groups of users with almost the same preferences, we can divide up the huge user-set into a significantly smaller number of subsets.

Another application of personalization and community building lies in networking. [5] If we take a distributed file sharing network for example. When a node is looking for a certain file, it queries it to its neighbors. If the neighbors do not have the file locally either, they request their neighbors, and so on. That way a request is propagated through the entire network.

It would be more performant if we grouped the users with the same interests in the same network-area. After all it is more probable that a neighbor, who has the same interests as the requester, will have the file locally stored. That way, less messages will be sent through the network and at the end it will work faster as you can see in 5.2.

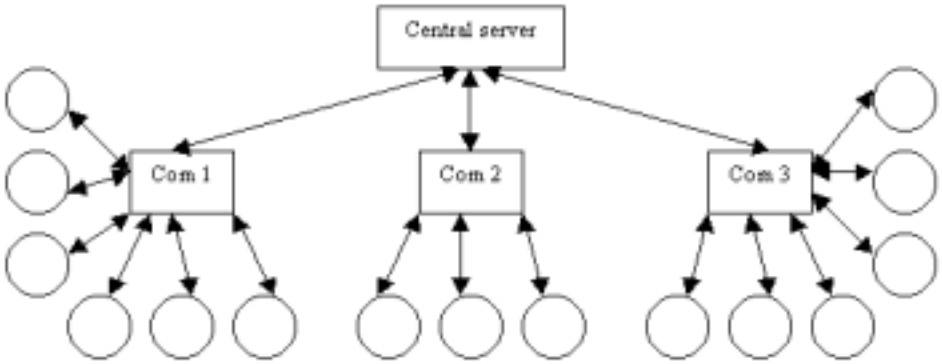


Figure 5.3: A centralised network

The same counts for a centralized file-sharing network which architecture is shown in 5.3. The users who have the same interests all connect to the same server. When a node looks for a file, it requests his Community-server for that file. If the file is not found on the Community-server, the request will be propagated through to the central server, which will query the other community-servers. That way, the number of requests sent through the network drops significantly, which will make the network work faster.

## 5.4 Conclusion

The Personal Television revolution will enable a new, closer, relationship between the broadcaster and viewer, and should result in more effective content management. However, business models are still unclear, and there are many process and data management issues to be resolved before the technology's full potential can be realized. The key advantages for the viewers are:

- Watch TV in the way they want, when they want
- Explore and acquire material across traditional broadcast and new on-line interactive services
- Combine the immediacy of television with the flexibility of the Internet

Also in networking there is a huge potential of personalization, in both a centralized and a distributed network.

# Part II

# The different methods

# Chapter 6

# Introduction

The tremendous potential of *digital television* is attracting interest from telecommunications providers, computer manufacturers, network providers, consumer electronic companies and broadcasters around the world. Pay Per View, high speed internet access, video on demand, cable telephony and e-commerce represent a portion of the new money spinning ventures in which industry firms are investing increasing amounts of dollars and resources [43].

**Digital television**, commonly known as digital TV, is a completely new way of broadcasting and is the future of television. It is a medium that requires new thinking and new revenue-generating business models. Digital TV is the successor to analog TV and eventually all broadcasting will be done this way.

Around the globe, cable, satellite and wireless operators are moving to a digital environment. In the US, all the major networks were slated to begin digital broadcasts in November 1999. By 2006, analog television will no longer be broadcast in the US. In Europe, the digital TV train is also rolling out of its station, with broadcasters in France, Ireland, Spain, Germany, Holland and the U.K. Most industry analysts are predicting that the transition to digital TV will be an evolution rather than a revolution, changing the way of life for hundreds of millions of families around the world.

In a world where digital TV is no longer a "far-from-my-bed show", *Personal TV* is within one's arm's reach. The first applications of Personal TV are already on the market(P VR, TiVo). Those applications use Personalization to recommend TV-programs.

**Personal TV** can best be described by the following:

- A service aspiration rather than a specific technology
- Allows viewing to be tailored to the individual
- Provides independence from the broadcast schedule
- Non-linear consumption of linear programs

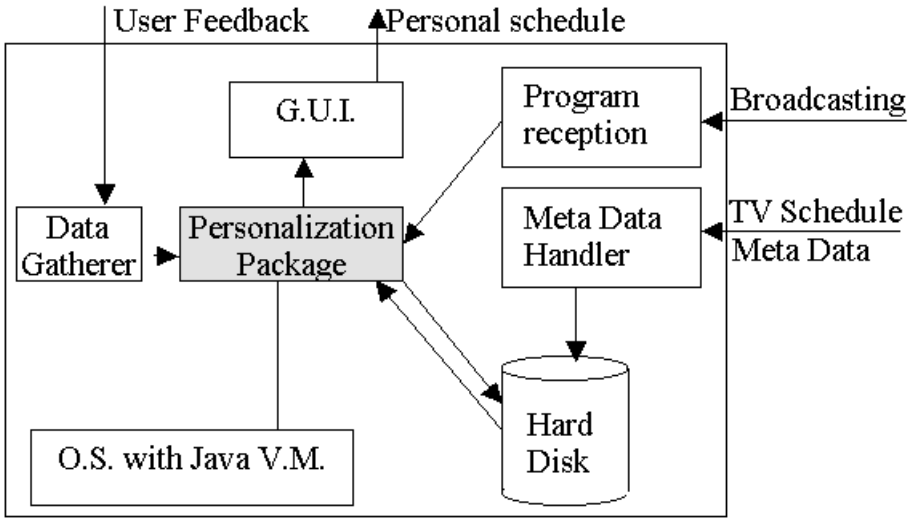


Figure 6.1: The local storage architecture on client side

- Automatic updating of program material
- Content from many sources
- Accurate tracking of media consumption

Personalization comes in two different architectures: *local storage* and *remote storage*. While the first stores the data of a TV program locally on a hard disk in the set-top box, the latter will only send the requested programs to the users[44].

## 6.1 Local storage

The broadcasters simply broadcast their TV-programs digitally. The *interesting* programs are stored locally on a hard disk in the set-top box or Personal Video Recorder. The only way we can separate interesting programs from others, is through meta-data. So each program must contain meta-data whether included in the file, or provided by an XML-file. As we can see in figure 6.1, we do not need **an uplink** in this architecture. That is, we do not need to send information from the client to the server. That simplifies the design of such a system.

It is clear that there are some variations on this architecture. For example, we can put all the program suggestion software on the server. In that case, the client has to provide the server with gathered data. After the necessary calculations have been done on server-side, the server sends a list with the TV program suggestions to the client.

| Year | Conservative | Aggressive |
|---|---|---|
| 2000 | 4 | 4 |
| 2005 | 40 | 256 |
| 2010 | 400 | 16.384 |

Table 6.1: Hours of video for $100

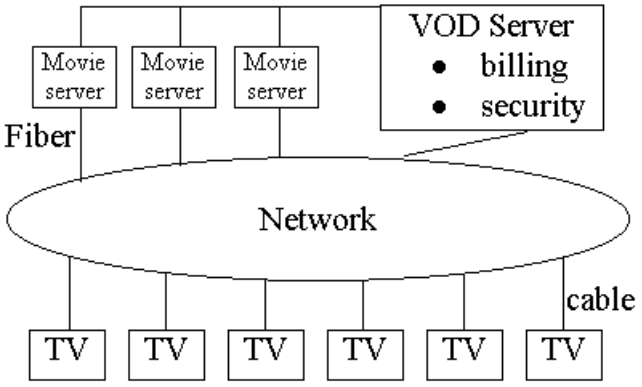


Figure 6.2: The remote storage architecture

This variation causes more data-transmission, but has the advantage that the set-top boxes don't have to be as complicated as before (less storage and processing power needed). Knowing that in computer-science, every piece of software is considered out-dated after half a year, this variation is worth considering. That way the suggesting software can easily be updated whenever needed, whereas in the other case, we would have to recollect all the set-top boxes. But this method does not guarantee the users privacy.

Disk drives enabling local storage, retrieval and manipulation of audio, video and data are increasing in capacity and falling in price. Future projections vary only as to the rate for the trends as you can see in table 6.1.

The conservative estimation, using an increase by factor 2 every 18 months, gives us up to 400 hours of video for only USD 100 in 2010. While the aggressive estimation, using an increase of factor 2 every 10 months, gives us more than 16.000 hours of video! We know that the real amount will be situated between those two extremes. That amount will certainly leave open the local storage option for mass consumption.

## 6.2 Remote storage

Figure 6.2 shows a possible architecture of a remote storage apllication. The users, when turning on their TV, will be showed a list of TV programs.

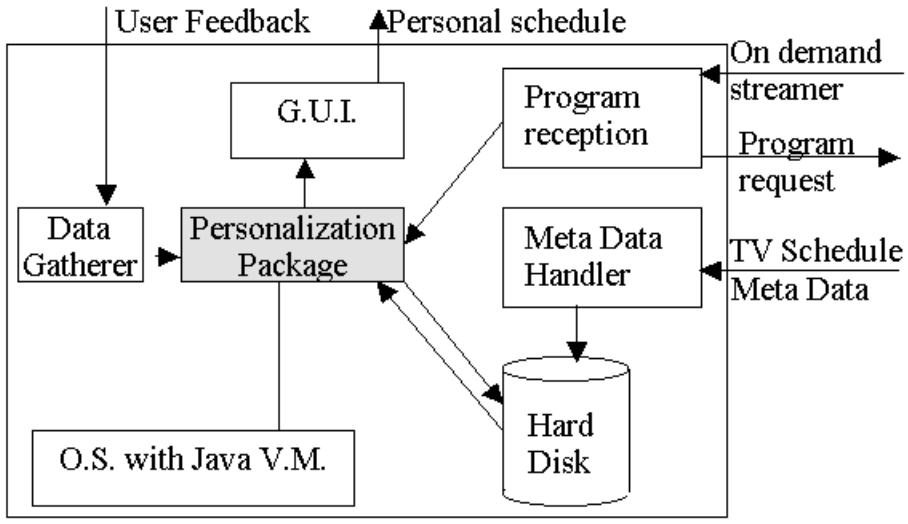


Figure 6.3: The remote storage architecture on client side

From this list, they choose one program. Then a request is sent over the network to the *video-on-demand* server. This server checks whether the user is allowed to see the requested movie. After the security, the billing and the movie selection information are done, the server streams the movie onto a video streamer. Then the video streamer streams the movie through the network to the set top-box, where it can be controlled by the VCR-functions like pause, play, rewind and so on. As we can see in figure 6.3, we need an uplink in this architecture because we do have to send information from the client to the server. This brings along a more complicated system design.

While the subscribers are watching the TV program, the TV is downloading the program from a *video-on-demand* server. For good quality, a bit rate from 1 to 20 mb/s is required. Knowing that cable allows bit rates up to 30.000 mb/s we see that only 1500 active users can be connected over one cable. There are two possible improvements to this system. The first is to decrease the necessary bit rate for good quality without loosing too much quality. This is being done with the new MPEG-4 standard, in which the necessary bit rate for a high quality movie is only 4 mb/s.

The second improvement lies in the transfer speed. Current fiber systems can handle terabits/s traffic within one fiber by using multiple optical wavelengths, each carrying up to 40 Gb/s traffic, thus achieving multi-terabit transmission. In the future, all-optical systems will handle greater than 1 terabit/s rates. With broadband predicted to be a reality within two years, we see that also this improvement is no longer a dream.

There is also a third architecture, which is in fact a combination of

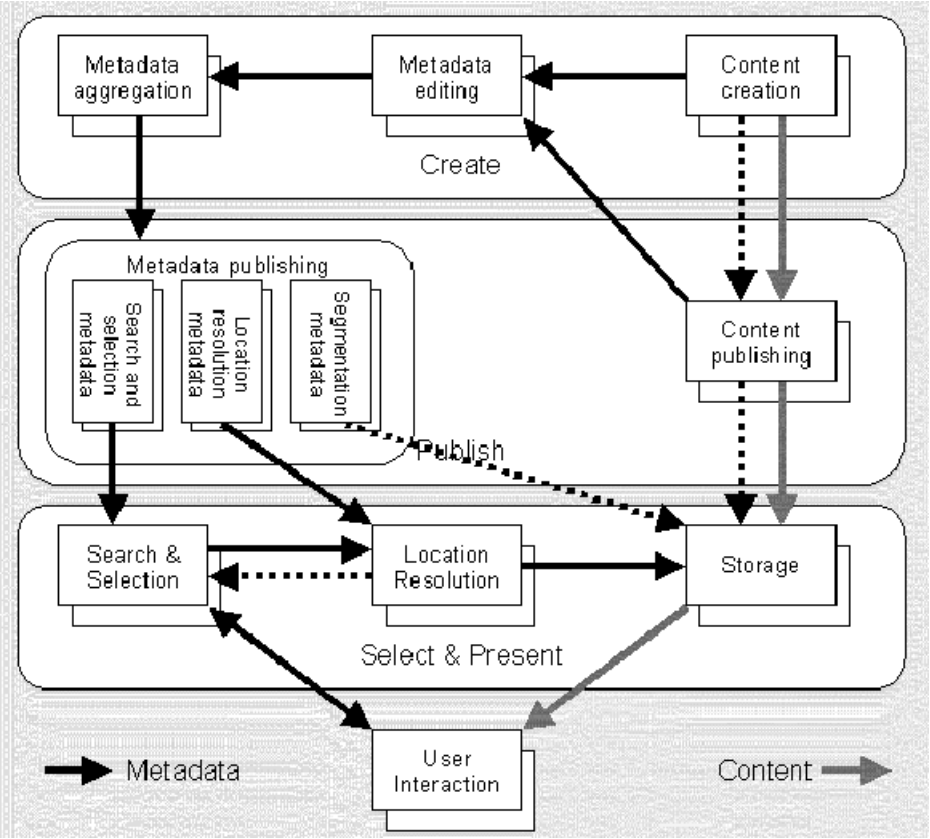


Figure 6.4: The way meta-data is used in Digital TV

both local and remote storage. It is called "near television-on-demand". It is clear that, when three million users are streaming a movie at the same time, the network will get overloaded and the quality will drop significantly. Therefore this third architecture was developed. It uses the same principle as the television-on-demand service. The only difference is that the movie is streamed to the client, which stores it locally. That way, the quality is ensured. The only drawback this architecture has, is that you always will have to wait some time before you can actually start seeing the requested TV program. But this architecture is clearly more scalable than the original television-on-demand.

## 6.3 The Meta Data

All business models that exploit the Personal TV technologies rely heavily on metadata. Currently, companies such as ReplayTV and Yes Television address this by compiling schedule data themselves. However, this is not

sustainable once content from multiple service providers becomes available to a device, and industry standards are required to ensure consistency. For example, when a viewer clicks on a program trailer, there will be the option to record all episodes of that series. How will the system know what programs belong to this series?

But things are getting better these days, and already two standards are rising (one in Europe and one in the USA)[48]. In the next years one standard will have to come out to survive. In figure 6.4 is shown how meta-data is composed and used in an application.

In this thesis, we assume that all the meta-data is provided in an XML-file with the following format:

```
<TRANSMISSION title="Friends">
  <KEYWORDS>
    <KEYWORD type="English" />
    <KEYWORD type="Series" />
    <KEYWORD type="Youth" />
    <KEYWORD type="Comedy" />
    <KEYWORD type="JENNIFER ANISTON" />
    <KEYWORD type="COURTENEY COX ARQUETTE" />
    <KEYWORD type="MATT LEBLANC" />
  </KEYWORDS>
</TRANSMISSION>
```

where 'English', 'Series', 'Youth' and 'Comedy' are the keywords of the TV Program 'Friends'. Knowing each TV program has exactly 7 keywords, would make it easier to make the necessary calculations. If that is not the case, we will have to take into account the number of keywords every program has, to adapt the scores in the user profiles. In this thesis the number of keywords worked with is considered as variable. You can see that the keywords of a TV program also include actor's names. In fact the Meta data should give a brief description of the program itself. After having read the Meta data of a program, you should know what the program is about.

Note that the station and the broadcast time are not mentioned in this Meta data. In a real Meta data file, they will certainly be included, but as they do not import for suggesting a program or not, they are not added in this example .

## 6.4 Different approaches

For composing a personalized program schedule, we have different possible solutions. In this part, we will investigate the main different solutions: the *individual solution*, the *collaborative solution* and the *community based*

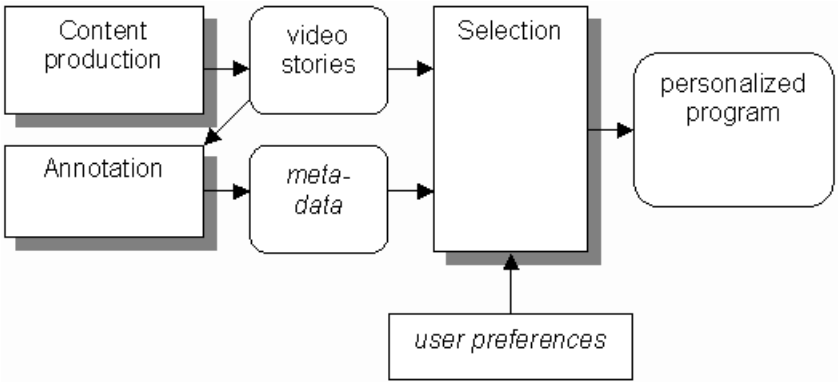


Figure 6.5: The Personalisation Process

*solution*. The *individual solution* only uses the personal profile for suggesting programs.

The first method, *the individual method* only uses the personal profile of a user. Matching the Program characteristics with the user profile, it will know whether the program has to be suggested or not.

The second method, *the collaborative method* is based on the assumption that the users with the same interests will have more or less the same opinion on a certain program. It will use that assumption to know whether to suggest a program or not.

Finally there is *the community-based method*. It makes use of community-detection. Groups of users with the same interests will be detected among the entire user-set. Knowing that a community contains all people with the same interests, a community profile can be composed. This profile will be used to recommend unknown programs with more certainty of succeeding.

We will test those methods by different criteria and discuss all the drawbacks and advantages they have. The criteria mentioned above are:

- The algorithm: what will happen during the execution?
- Performance: is the system scalable and fast enough?
- Architecture: which modules are needed on which sides?
- Data and data transfer: which data is necessary?
- Ethical aspects: is the privacy guaranteed?
- Functionality: is it method appreciated by the users?
- Possible improvements

In all three solutions, we have to know that personalization is in fact an iterative process. The user will be able to rate the programs that were suggested continuously. That way every cycle, the users profile will get better (and so will the suggested programs).

In figure 6.5 an external architecture is shown. We assume here that the Meta-data comes from an external server. The Program suggestions software will be located in the Selection module of the architecture, which will be located in the set top box of the client.

At the end of this part, the three methods will be compared using an inquiry. The conclusions will be drawn from the inquiry and the best approach of suggesting programs will be stated.

# Chapter 7

# The Individual Method

As we already know the client needs a set-top box[45] if he wants to join this program suggestions service. In the set-top box, we will have to store the users profile. A user profile will be composed from the moment the user rates certain programs. From that moment on, the profile will be adapted continuously.

In the user profile, there will be data storage of what preferences that user has. The form of the data-tuples is the following:

```
Soap_opera 486
```

Where 'Soap_opera' is the keyword and '486' is the score related to the keyword 'Soap_opera'.

In this chapter we will discuss how we can recommend programs only using the individual profile of a user. This way of suggesting programs is called **content based filtering**[19]. It takes in the User profile from the user and the XML TV program schedule. Every TV-program will be compared to that profile. A score will be given to each program to see whether the user will appreciate the program or not.

## 7.1 The algorithm

This method is the simplest method of them all. It takes the meta-data from a TV-program-schedule and a User Profile as input. It outputs a score telling the user whether this TV-program is suggested or not. At that point, the user still can choose which TV-programs he watches.

When a user starts up the system for the first time, his opinion is asked on 50 TV-programs randomly picked out of the TV schedule. The user can rate the programs with one of the following rates: 'not known', 'very bad', 'bad', 'neutral', 'good' or 'very good'. When the user chooses 'very bad' for example, all the keywords from that program get -2 on their score in the user profile. Similarly the keywords get adapted with scores from -2 to +2

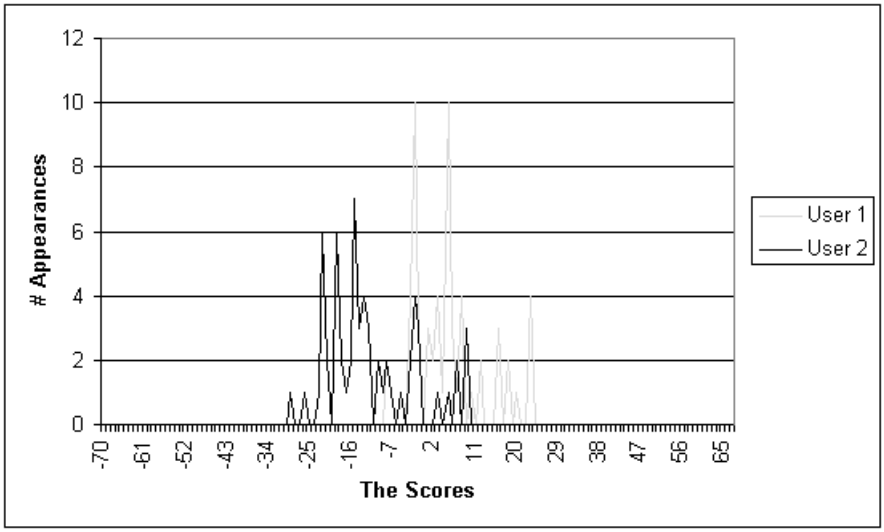


Figure 7.1: The scores distribution of 62 programs, given by 2 different users

depending of what the rate of the program was. That way, an initial user profile is created.

### 7.1.1 Scanning the User Profile

When the Program Suggestions-method is called, first a scan is made in the User Profile to see what the average score of that person is. Because people have **different taste** but also **different valuation**, we have to take into account the kind of scores the user gave. This scan results in a threshold (= the *user-threshold*), which is used to adapt the program-score.

#### Why?

Imagine having two users. The first user easily gives a 'very good'-rate while the other-one's highest rate is 'good'. This would result in the first person being suggested a lot of TV-programs, while the second user only would get a very low number of recommended programs. That is not what we really want, so we will take into account the valuation of the users, in terms of a threshold. The user-threshold of the former will be approximately +3, while the user-threshold of the latter will be -1.

#### How?

A loop is started on a fixed number of TV programs. For each TV program we calculate the score that TV program would get from the user. All the resulting scores are stored in a vector.

As we see in figure 7.1, the scores a program can get in this example, are always situated between -68 and +69. These are not fixed numbers, but are related to this example. We can also see that the two users have a different validation for the programs. The first user easily gives a high rate to a program, while the second is a lot more critical on rating TV programs.

We define $P$ as the set of TV programs that are included in the Program schedule from one week.

$$P = \{p_i = TVprogram \in ProgramSchedule\} \qquad |P| = p$$

Knowing that $x_i$ is the score of a program $p_i$ given by a user U, then

$$ut = Average(X) = \frac{\sum_{i=1}^{p}(x_i)}{p}$$

is the *userthreshold* for that user U.

We can easily calculate the Average(X) for the users from the example. The first user will have an average of 2.15 and the second user will have an average of -5.15. That value is outputted as the user-threshold by the UserProfileScan. Here it becomes clear again that the first user easily gives high quotations for TV programs while the second is very critical. It also means that we will have to be a little more critical recommending programs to the first user. For the second user, we will use the standard recommending procedure because we don't want to suggest programs he might dislike.

### 7.1.2 Rating the TV programs

After obtaining the user-threshold, the procedure loops on the keywords of the program meta-data. For each keyword, it looks up the score in the user profile of the user. It adds all the scores of the keywords, resulting in a SUM-score.

Knowing that $p_i$ is a TV program with $p_i$.MetaData being the meta-data associated with P.

$$K_i = \{k_j = keyword \in p_i.MetaData\} \qquad |K_i| = k$$

Let $UP$ be a User Profile of a person with user-threshold $ut$ **bigger** than 0 and #K = k. Then,

$$\begin{aligned} Score(p_i) &= \sum_{j=1}^{k}(UP.getScore(k_j)) - ut \\ \text{OR} & \\ Score(p_i) &= \sum_{j=1}^{k}(UP.getScore(k_j)) - \frac{\sum_{i=1}^{p}(x_i)}{p} \end{aligned}$$

| Rate | Percentage | Interval |
|---|---|---|
| 'very bad ' | 10% | $[m, m + \|M - m\| * 0.1]$ |
| 'bad' | 20% | $[m + \|M - m\| * 0.1, m + \|M - m\| * 0.3]$ |
| 'neutral' | 40% | $[m + \|M - m\| * 0.3, m + \|M - m\| * 0.7]$ |
| 'good' | 20% | $[m + \|M - m\| * 0.7, m + \|M - m\| * 0.9]$ |
| 'very good' | 10% | $[m + \|M - m\| * 0.9, M]$ |

Table 7.1: The distribution of the interval

is the score the program gets from that user.
Let $UP$ be a User Profile of a person with user-threshold $ut$ **lower** then 0 and #K = k. Then,

$$Score(p_i) \quad = \quad \sum_{j=1}^{k} (UP.getScore(k_j))$$

is the score the program gets from that user.

### 7.1.3 Suggesting a TV program

Now the question rises how we know when a program $p_i$ with its score $Score(p_i)$ is classified under a certain category. A program can be classified in one of the following five categories: 'very bad', 'bad', 'neutral', 'good' or 'very good'.

#### The distribution of the interval

We know what scores the TV programs in the XML schedule have, so we know the minimum $m$ and the maximum $M$ score. This leaves us with an interval of $[m, M]$ in which the program-valuations are classified. If we now split up this interval in 5 peaces. We want only 10% of the TV programs to be classified in the 'very good' category, 20 % of the TV programs to be classified in the 'good' category, 40 % in the 'neutral' category. Similarly we want 20 % in the 'bad' category and 10 % in the 'very bad' category. The result of these calculations can be seen in table 7.1.

#### The Classification

Now we know the boundaries for classifying the TV programs in, we can start the classification process. The only thing that has to happen is comparing the Score of the program P with the boundaries of the intervals. If the suited interval is found, the rate of the program is known, and outputted.

## 7.2 Performance

In this method, three things are in fact happening:

- Scanning the User Profile
- Calculating the classification borders
- Looping over the TV programs and giving scores

### 7.2.1 Scanning the UserProfile

The performance of scanning the User Profile is totally independent of the amount of users that joined the service. Because the scanning only needs the personal profile, it can execute locally on the set-top box of the client. It only depends on how detailed the profile is (how much keywords are stored in the profile).

If we make the scanning-loop with a fixed number of p TV Programs and k is the number of keywords characterizing a Program, every loop-cycle, the following pseudo code is executed:

```
∀p_i ∈ P {
Score(p_i) = 0
   ∀k_j ∈ K_i {
      Score(p_i) = Score(p_i) + UP.getScore(k_j)
   }
   //Save the Score in a vector
}
```

where $UP = \{k_up = keyword \in UserProfile\}$ and $|UP| = n$.

The getScore()-method has a performance of $O(\log(n))$, because this method uses a binary search algorithm to find the score of keyword K. Knowing this and the fact that there are k keywords in the meta-data characterizing a TV Program, it is easy to calculate the Complexity of the process.

$$\begin{aligned} T(p,k,n) &= p.k.\log(n) \\ \rightarrow O(ScanUserProfile) &= O\Big(p.k.\log(n)\Big) \end{aligned}$$

We see that this function contains both a quadratic and a logarithmic part. The only question that remains is which part will preponderate. This depends on the exact values of p, k and n. We know that p is the number of programs in the week-TV-schedule. We also now that k is the number of keywords specifying a program. Those numbers are variable and can be chosen by the service provider. So they can in fact decide which part will weigh through the most.

### 7.2.2 Calculating the classification borders

Just as the previous part, calculating the borders is independent of the amount of users that joined the personalization service. Its execution also happens locally on the client's set-top box. For calculating the borders we need the minimum and the maximum score that has been given to a program. Those numbers are already known after the scan of the user Profile is done.

It takes four steps for obtaining the four boundaries. It are four calculations of the following form:

$$Boundary(p\%) = m + |M - m| * p\%]$$

where $M$ is the Maximum and $m$ the minimum score and $p$ is the percentage of scores we want to be in the segment.

### 7.2.3 Looping over the TV programs and giving scores

This procedure will loop over all the TV programs, which are offered in the XML schedule. A score has to be calculated for each program, telling the user whether this program is suggested or not.

```
∀p_i ∈ P {
   int score = 0;
   ∀k_j ∈ K_i {
      score = score + UP.getScore(k_j);
   }
}
```

The performance of this procedure depends on three things: the order of the getScore-method, the amount of keywords characterizing each program and the number of programs to be processed. Knowing the amount of keywords is k, the amount of programs included in the XML-schedule is p and that there are n keywords in the User Profile UP, we see that

$$\begin{aligned} T(p,k,n) &= p.k.\log(n) \\ \rightarrow O(ScanUserProfile) &= O\Big(p.k.\log(n)\Big) \end{aligned}$$

is the complexity of the Looping section.

It is possible that we include this section in the scanning section. Because the same pocess is happening in both parts. That would be an optimization, that would result in a better performance. Because it would reduce this section to a retrival of the correct program score. That would result in an operation of $\log(p)$, if a binary search method would be used.

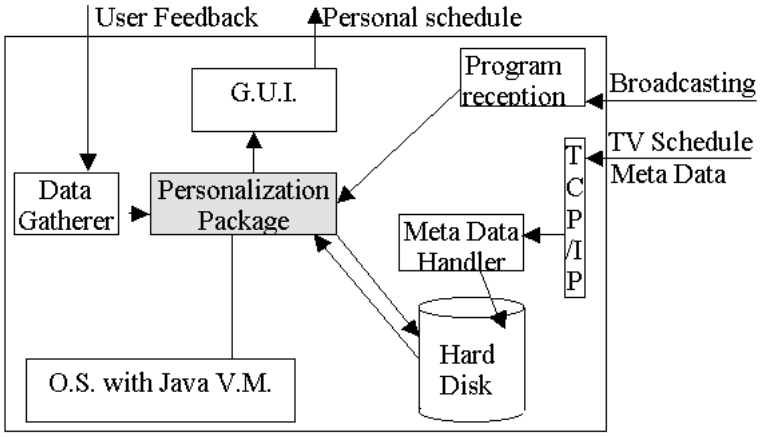


Figure 7.2: The Individual method architecture

### 7.2.4 Conclusion

The total execution time of this method is

$$\begin{aligned} T(p,k,n) &= p.k.\log(n) + 4 + p.k.\log(n) \\ &= 2.p.k.\log(n) + 4 \\ &\downarrow \\ O(ProgramSuggestions) &= O\Big(p.k.\log(n)\Big) \end{aligned}$$

## 7.3 Architecture

In the previous section, we could see that for suggesting programs, this method doesn't need to connect to a server. The entire program suggestions process can execute locally on the clients set-top box. We only need a connection from the server to the client. In figure 7.2 you can see that we don't need an uplink to the server. Knowing this, we see that this method can be used in a **local storage** or a **remote storage** architecture.

In this architecture we need a bit of processing power and local storage possibilities on the set-top box. As we could see in figure 7.2, we need to calculate everything locally. This ensures the privacy of the user but results also in the need of local processing power and storage.

## 7.4 Data and data transfer

On the Hard Disk of the set top-box, the entire TV-schedule for one week is stored in XML format. Every days television schedule has a separate file identified with an exclusive number. That number identifies the file. When a day is done, the XML file that contained the television schedule of that day can be removed from disk.

The only data transfer that has to take place in this method, is the providing of new TV-schedules to the client. This can be done day by day. Every morning, for example, the set top box could make a connection with the server, requesting an update on the television schedule for the coming week. At that time, the set top box compares the file-ID's it has on its hard disk with the file-ID's on the server. If the files-ID's that overlap remain the same, and the files with new ID's are downloaded from the server to replace the older files.

If a change to the television schedule is made on a certain day, the server's file of that day will be renamed. When the client requests the server for an update on the television schedule, the set top box will notice the change, and it will update the file as explained above.

## 7.5 Ethical aspects

This method doesn't really cause ethical problems. Every calculation is done locally and no personal data is sent out to the server. That way the privacy of the user is assured. We only have to make sure that the user really is aware of that. Because otherwise, he might dislike the product for that ungrounded reason.

Still, the relation between the user and the software remains. Therefore, the writer of the software must really stay objective writing the algorithms for the program suggestion. He must really keep in mind the goal he has. That goal is the providing of a service to the users; making personal TV program suggestions. And not, for example misuse te collected user-data for personalized publicity or for other goals.

The way the knowledge is modeled is also a decision the programmer has to make. Making that decision, the program is in fact deciding how the users are thinking. It is not correct to say that all the users think the same. We would have to know at least some other user characteristics for knowing how a certain user will react. So, there the need of a **qualitative investigation**[1] is stated.

But a qualitative investigation brings along privacy issues. We have to take into account the possible consequences this investigation has on privacy level. And even if we would know some user characteristics, we would never

[1] An investigation that looks for qualitative information like gender, age, ...

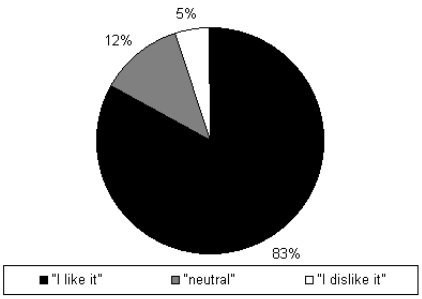


Figure 7.3: The appreciation of method 1, by 200 users

been able to create a new knowledge modeling for each user. These two facts show us that it is better to restrict our investigation to a quantitative level.

A **quantitative investigation**[2] also has the befit that every user is treated equally. It does not depend who you are, but only what kind of TV programs you are interested in. Keeping the primary goal in mind, it is obvious that is possible to realize that goal without having to know how old a certain person is.

This individual method will never perform as good as a collaborative method from ethical point of view. Imagine we have two situations. In the first situation, we have only one person giving his opinion on TV programs. In the second situation, there are 500 users giving their opinions on the same programs. It is clear that in the second situation we have a lot more information on the appreciation of the programs. So from this point of view, the collaborative and community-based solutions will provide better program suggestions than this method.

Because of the need of meta-data, the user will have to be subscribed to a service, which provides the meta-data. This may bring along an additional cost for the user, which is a disadvantage worth mentioning.

## 7.6 Functionality

This method recommends the programs the users like and disapproves the programs the users dislike. In general, this method is approved by 83% of the users. It means that 83% of the users liked the programs that were suggested. An inquiry was done to state this number as you can see in figure 7.3. The investigation was done on a randomly picked group of 200 persons. They were asked to rate 50 TV Programs. After that was done, they were

[2] An investigation that looks for quantitative information like how high a TV program is rated

sent both a list of suggested TV programs and a list of disapproves TV programs. They were asked their opinion on the recommendation. They could choose between 'I like it', 'neutral' or 'I dislike it'.

## 7.7 Possible improvements

We already mentioned that program suggesting is an iterative process. At any time, a user can rate a program. Doing that, the user gets a more and more specified User Profile. The program, knowing the users preferences will be better at suggesting new TV programs to the user. Some times, it can be a good thing to recommend a new type of program to a user. That way the user will not come to a standstill in his television watching behavior. The program has to help the user to discover new programs.

Each iteration, the program suggestion will get a better appreciation. This, because every time the user gives us some feedback on a program whether he liked it or not, the program will know the user better. That way, the suggested programs will probably please the user more and more.

# Chapter 8

# The Collaborative Method

This method will use a combination of **content based** and **collaborative filtering** for suggesting TV programs. For content based filtering, we need to match the program characteristics with the user profile. The user profile will be composed the same way it was done in the previous method.

Collaborative filtering [46] can be defined by "'Guiding people's choices of what to read, what to look at, what to watch, what to listen to (the filtering part); and doing that guidance based on information gathered from some other people (the collaborative part)"'. It is obvious that we need the profile of a certain number of users for doing that.

In this chapter we will discuss how we can recommend programs using **both** the individual profile of a user and the profiles of the other users. We will discuss what consequences this method has.

## 8.1 The algorithm

This method takes the meta-data from a TV-program-schedule, a UserProfile and the set of UserProfliles of the other users as input. It will combine the individual method (described in the previous chapter) with the general opinion the other users have on a certain program. It will output a score telling the user whether this TV-program is suggested or not. At that point, the user still can choose which TV-programs he watches.

When a user starts up the system for the first time, his opinion is asked on fifty TV-programs randomly picked out of the TV schedule. The user can rate the programs with one of the following rates: 'not known', 'very bad', 'bad', 'neutral', 'good' or very good'. When the user chooses 'very bad' for example, all the keywords from that program get -2 on their score in the UserProfile. Similarly the keywords get adapted with scores from -2 to +2 depending of what the rate of the program was. That way, an initial UserProfile is created.

After that is done, the server is requested for the *UserProfileSet*. This

is the set of all[1] the User Profiles from the users who joined the service. The server sends the UserProfileSet to the client, which stores it on its hard disk. This is done, because the collaborative method takes into account the opinion of the other users on a certain program.

### 8.1.1 Scanning the UserProfileSet and the UserProfile

When the ProgramSuggestions-method is called, first a scan is made in the UserProfileSet to see what the average score of the persons is. If the general opinion on the TV programs is higher then 0, we have to take that into account suggesting a program or not. This scan results in the *userSet-threshold*. The same is then repeated in the UserProfile from the user which called the procedure. This scan results in the *user-threshold*. Both the thresholds are stored and propagated through to the rest of the method.

#### Why?

The reason for obtaining the user-threshold became clear in the previous chapter. If you want to know more on that I would refer to section 7.1.1. The reason we want the userSet-threshold is in fact the same as the reason of why we wanted the user-threshold. We want to know with what kind of people we are dealing. It can possibly be that the general opinion on the programs is very high. This could mean two things: The users are pleased with the television schedule or the users easily give good rates. Because we depart from the vision that the users **only** want *good* programs to be suggested and the fact that the user will prefer less suggested programs but all good ones above the suggestion of a lot of programs where some ad are included. Wanting to suggest only the relative best programs, we have to take into account how the general opinion is. If it is 'good', we should be more critical suggesting a program than if it were 'neutral'.

#### How?

For how the calculation of the user-threshold, please refer to 7.1.1. Just like in the previous chapter, we define P, K, UP and UPS as follows:

$$\begin{array}{rclrcl} P & = & \{p_i = program \in Televisionschedule\} & |P| & = & p \\ K_i & = & \{k_j = keyword \in p_i.MetaData\} & |K_i| & = & k \\ UP & = & \{k_up = keyword \in UserProfile\} & |UP| & = & n \\ UPS & = & \{UP_s = program \in UserProfileSet\} & |UPS| & = & m \end{array}$$

The following pseudo-code is used for calculating the userSet-threshold.

```
int validationScore = 0; {
∀p_i ∈ P {
```

[1] It can be a randomly picked set of User Profiles

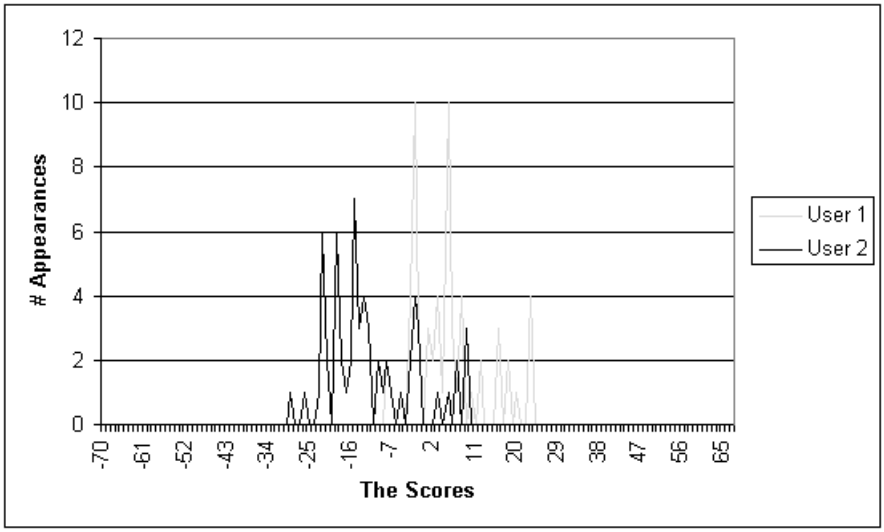


Figure 8.1: The scores distribution of 15 programs, given by 375 users

```
    ∀UP_s ∈ UPS {
        ∀k_j ∈ K_i {
            validationScore = validationScore+UP_s.getScore(k_j)
        }
        //Save the validationScore in a vector
        //NumberVector.incremet(validationScore)
    }
}
```

A loop is started on a fixed number of TV programs. For each TV program we calculate the score that TV program would get from the different users. For doing that, we have to take a look at the keywords of the program and look up the score of those keywords in the User Profile. The sum of all these scores is stored in a vector. Also the number of times each score appears is kept up to date in the NumberVector.

As we see in figure 8.1, the scores a program can get in this example, are always situated between -68 and +69. We can also see that this graph follows a **Gaussian distribution**. After having done this experiment several times with single users, we notice that the scores distribution **always** follows a Gaussian distribution.

For obtaining the average appreciation score of the programs, we now have to calculate E(X). This can be done like this:

Knowing that X is the validationScore a program gets from a user U and

the number of programs rated by the user is m, then

$$ust = E(X) = \sum_{i=1}^{p}(x_i.P(X = x_i))$$

is the expectation value for X from a certain user U. As you can see, we need to know the probability of $X = x_i$. This can be calculated by solving the following equitation.

$$P(X = x_i) = \frac{\#\text{ of measures where X} = x_i}{\#\text{ of measures done}}$$

The number of measures done, is equal to *#Programs.#UserProfiles (= p.m)*. So we only have to know the number of measures where X equals $x - i$. That amount can be retrieved in NumberVector on index $x_i$.

In the example of figure 8.1, the value of E(X) is 3.03895. In this example, the users clearly gave good validations to the programs. Therefore we will have to be more critical in suggesting programs to users from that user Set. The value of *E(X)* will be outputted as the userSet-threshold *ust*.

### 8.1.2 Rating the TV programs

After obtaining the he user-threshold and the userSet-threshold, the procedure first calculates the personal score by looping over the keywords of the program meta-data. For each keyword, it looks up the score in the user Profile of the user and adds it to the PersonalScore.

After that is done, it loops over the other user Profiles that are in the UserProfileSet and looks up the scores the other users would give to the TV program. It adds all the scores of the other users and divides it by the number of UsersProfiles that are in the UserProfileSet $UPS$. That results in the UsersScore. Both the scores are then added and divided by 2 to obtain a new adapted score for the program $p_i$. This is all resumed in the following pseudo-code:

```
∀p_i ∈ P {
   int PersonalScore = 0 {
   int UsersScore = 0 {
   ∀k_j ∈ K_i {
      PersonalScore = PersonalScore+PUP.getScore(k_j)
   }
   ∀UP_s ∈ UPS {
      ∀k_j ∈ K_i {
         UsersScore = UsersScore+UP_s.getScore(k_j)
      }
```

```
    }
    UsersScore=UsersScore/UserProfileSet.size()
    Score(p_i)=( (UsersScore-ust)+(PersonalScore-ut) )/2
}
```

where PUP is a User Profile of a certain user with user-threshold $ut$ and $n$ as the number of programs rated by the user. As defined before, $m$ is the number of user Profiles in the UserProfileSet UPS.

$$\begin{aligned} Score(p_i) &= \frac{\sum_{j=1}^{k}(PUP.getScore(k_j)) - ut}{2} \\ &\quad + \frac{\frac{\sum_{s=1}^{m}(\sum_{j=1}^{k}(UP_s.getScore(k_j)))}{m} - ust}{2} \end{aligned}$$

OR

$$\begin{aligned} Score(p_i) &= \frac{\sum_{j=1}^{k}(PUP.getScore(k_j)) - \frac{\sum_{i=1}^{p}(x_i))}{p}}{2} \\ &\quad + \frac{\frac{\sum_{s=1}^{m}(\sum_{j=1}^{k}(UP_s.getScore(k_j)))}{m} - \sum_{i=1}^{p}(x_i.P(X=x_i))}{2} \end{aligned}$$

is the score that program $p_i$ gets from that user if he had a $ut$ bigger than 0. If the $ut$ or the $ust$ are smaller than 0, we don't take them into account for calculating the program score. The reason became clear in the previous chapter.

### 8.1.3 Suggesting a TV program

This raises the question how we know when a program $p_i$ with its score Score($p_i$) is classified under a certain category. A program can be classified in one of the following five categories: 'very bad', 'bad', 'neutral', 'good' or 'very good'.

#### The Gaussian distribution

We know that the graph of the X = score($p_i$) is always following a Gaussian distribution $N(\mu, \sigma)$, N(3.03895,16.377088) in the example. When we now know that we want only 10% of the TV programs to be classified in the 'very good' category, we can calculate easily that:

$$\begin{aligned} P\left(X \leq \frac{Score(p_i) - \mu}{\sigma}\right) &= 10\% = 0.1 \\ &\Updownarrow \text{ using the statistical tables} \\ 1.28 &= \frac{Score(p_i) - \mu}{\sigma} \end{aligned}$$

| **Rate** | **Percentage** | $\frac{Score(P)-\mu}{\sigma}$ |
|---|---|---|
| 'very good' | 10% of the programs | 1.28 |
| 'good' | 20% of the programs | 0.53 |

Table 8.1: Normal Distribution Function

| **Rate** | **Percentage** | **Scores** |
|---|---|---|
| 'very good' | 10% of the programs | $24 < Score(P)$ |
| 'good' | 20% of the programs | $12 < Score(P) \leq 24$ |
| 'neutral' | 40% of the programs | $6 < Score(P) \leq 12$ |
| 'bad' | 20% of the programs | $-18 < Score(P) \leq -6$ |
| 'very bad' | 10% of the programs | $Score(P) \leq -18$ |

Table 8.2: Program Scores Classification

$$\begin{array}{rcl} & \Updownarrow & \\ Score(p_i) & = & (1.28.\sigma) + \mu \\ & \Updownarrow & \\ Score(p_i) & = & 24 \end{array}$$

**The Classification**

Similarly the following boarders can be calculated for the good and 'neutral' rates. For the 'bad and 'very bad' rate, we can make use of the symmetrical property of the Gaussian distribution. Knowing that $E(X) \approx 3$, it is easy to calculate that for being categorized in the 'very bad' category, a program needs to have a score of:

$$\begin{array}{rcl} Score(p_i) & = & E(X) - (24 - E(X)) \\ & \Updownarrow & \\ Score(p_i) & = & 3 - (24 - 3) \\ & \Updownarrow & \\ Score(p_i) & = & -18 \end{array}$$

On overview of the results of the calculations is given in table 8.2.

## 8.2 Performance

In this method, four things are in fact happening:

- Scanning the UserProfile
- Scanning the UserProfileSet
- Calculating the classification borders
- Looping over the TV programs and giving scores

### 8.2.1 Scanning the UserProfile

The performance of scanning the UserProfile is totally independent of the amount of users that joined the service. Because the scanning only needs the personal profile, it can execute locally on the set-top box of the client. It only depends on how detailed the profile is (how much keywords are stored in the profile). Earlier in this chapter, we defined this number as $n$.

The same pseudo-code is executed as in the previous chapter for obtaining the user-treshold. Therefore it is easy to see that the same execution time will be needed here.

$$\begin{aligned} T(p,k,n) &= p.k.\log(n) \\ \rightarrow O(ScanUserProfile) &= O\Big(p.k.\log(n)\Big) \end{aligned}$$

Just like in the previous chapter, we see that it consists of two parts; an exponential and a logarithmic part. But because the service provider can decide the values of $p$, $k$ and $n$, it is not a problem. The advantages off incrementing them should be compared with the loss of performance in terms of execution speed.

### 8.2.2 Scanning the UserProfileSet

As you could see in section 8.1.1, the scanning of the UserProfileSet is a bit more complex. this results in a longer computation time. If $p$ is the number of programs looped on, $k$ the number of Keywords in the meta-data of one TV program. Let $n$ be the number of keywords in the meta-data of the user Profile and $m$ the number of UserProfiles in the UserProfileSet.

$$\begin{aligned} T(m,p,k,n) &= m.p.k.\log(n) \\ \rightarrow O(ScanUserProfileSet) &= O\Big(m.p.k.\log(n)\Big) \end{aligned}$$

It is obvious that this computation will weigh through more than the previous one. The crucial factor is $m$, and therefore we really need to determine the best value of $m$. This has to be done on experimental bases. Unfortunately I didn't have the required resources for doing those experiments.

### 8.2.3 Calculating the classification borders

Just like in the previous part, the calculating of the borders is independent of the amount of users that joined the personalization service. Its execution also happens locally on the client's set-top box. For calculating the borders we need the statistical value which can be found in the table of Gauss (table 8.1 shows an extract of the needed values). But because we only need the values for 10% and 30% we don't need the entire table. The only thing we have to do is store these variables and do an equitation of the form shown in:

$$1.28 \quad = \quad \frac{Score(p_i) - E(X)}{V(X)^2}$$

The only calculation that has to take place is the calculation of $\sigma$ and $\mu$.

We know that:
$$\begin{array}{lllll} \sigma^2 & = & V(X) & = & E(X^2) - \Big(E(X)\Big)^2 \\ \mu & = & E(X) & = & \sum_{j=1}^{m}(x_j.P(X = x_j)) \end{array}$$

The calculation of $E(X)$ and $E(X^2)$ have already happened in the previous step. So the five calculations that have to be done in this step, are very small and will go very fast.

### 8.2.4 Looping over the TV programs and giving scores

This procedure will loop over all the TV programs, which are offered in the XML schedule. A score has to be calculated for each program, telling the user whether this program is suggested or not. The way the score of a program $p_i$ is calculated, can be seen in section 8.1.2. We see it consists of two parts: calculating the PersonalScore and calculating the UserScore.

The performance of this procedure depends on four things: the order of the getScore()-method, the number of User Profiles $m$ in the UserProfileSet, the amount of keywords $k$ characterizing each program and the number of programs $p$ to be processed.

$$\begin{array}{rcl} T(m,p,k,n) & = & m.p.k.\log(n) \\ \rightarrow O(Looping) & = & O\Big(m.p.k.\log(n)\Big) \end{array}$$

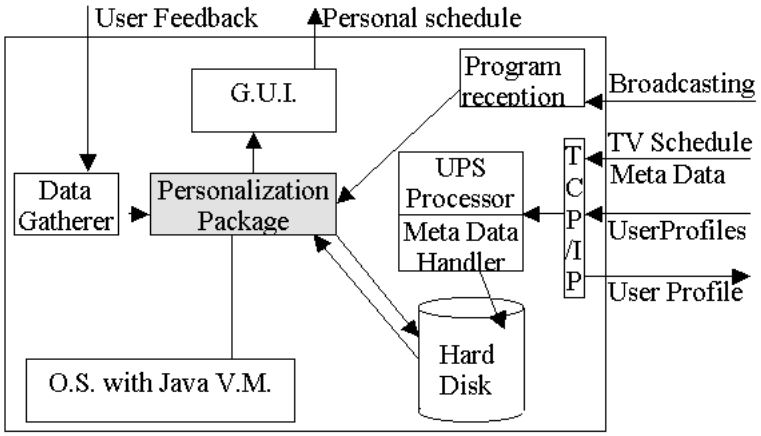


Figure 8.2: The Collaborative method architecture

But those calculations are already done simultaneously with the calculation of the UserSet-threshold and stored in a vector. That way, we don't have to take into account this product but only the retrieval time of the correct program score, which is equal to $p$.

### 8.2.5 Conclusion

The total performance of this first method is the sum of the order

$$\begin{aligned} T(m,p,k,n) &= p.k.\log(n) + m.p.k.\log(n) + 5 + m.p.k.\log(n) \\ \rightarrow O(ProgramSuggestion) &= O\Big(p.k.\log(n) + m.p.k.\log(n) + 5 + m.p.k.\log(n)\Big) \\ &= O\Big((2m+1).p.k.\log(n) + 5\Big) \\ &= O\Big(m.p.k.\log(n)\Big) \end{aligned}$$

The only remaining question is which part will weigh through the most; the first (exponential) part or the second (logarithmic) part. The service provider can give the answer on that question. Because $m$, $p$, $k$ and $n$ are variable and can be chosen by the service provider. It is clear that, the higher those values are, the better the result will be, but also the longer the procedure will take to execute.

## 8.3 Architecture

There is not a big difference between this architecture and the architecture of the previous method. There are only two new elements on the client side.

The first is an addition of a module that can cope with the receiving of a UserProfileSet. The second is a module that can send the personal User Profile to the server. as we can see here, we **do** need an uplink to the server to use this method. Therefore only the **remote storage** is considered here. We could also use a **local storage** architecture, but then the necessary changes must be made to obtain an uplink to the server.

On the server side, things get more complex now. Because now the amount of data send to the server is a lot bigger than in the first method, we have to take hard- and software precautions. The architecture of the clients set-top box is shown in figure 8.2.

## 8.4 Data and data transfer

On the Hard Disk of the set top-box, the entire TV-schedule for one week is stored in XML format. Every day's television schedule has a separate file identified with an exclusive number. That number identifies the file. When a day is done, the XML file that contained the television schedule of that day can be removed from disk. Also the UserProfileSet has to be stored locally on the hard disk.

There are 3 types of data transfer that have to take place in this method. The first is the acquisition of new TV-schedules by the client. The second is the acquisition of the UserProfileSet. The third is the sending of the personal UserProfile to the server, which collects all the User Profiles. These transfers can be done day by day. Every morning, for example, the set top-box could make a connection with the server, requesting an update on the television schedule for the coming week and on the UserProfileSet. At that time, the set-top box compares the file-ID's it has on its hard disk with the file-ID's on the server. The files-ID's that overlap remain the same, and the files with new ID's are downloaded from the server to replace the older files. The UserProfileSet has to be replaced entirely.

If a change to the television schedule is made on a certain day, the server's file of that day will be renamed. When the client requests the server for an update on the television schedule, the set top-box will notice the change, and it will update the file as explained above.

For sending the personal User Profiles to the server, the server needs an architecture that is *scalable*. That means the server must be able to cope with a lot of client-connections at the same time. Therefore we must assure that not too much data has to be sent through at the same time.

## 8.5 Ethical aspects

This method causes ethical problems including those that were mentioned in the previous chapter. Because the personal User Profile is sent out to

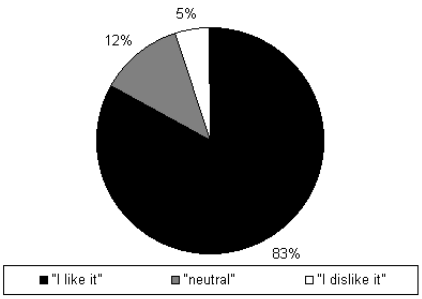


Figure 8.3: The appreciation of method 1, by 200 users

the server, a privacy policy should certainly be made up. But if the user should send his profile to the server, without an ID coupled to the profile, it should not cause an ethical drawback. But the user should be aware of that and the company should well protect their data-connection and secure their server for unwanted guests.

For the service-provider, the data they gain, is very important. That data can be solved to the broadcasters because it is a lot more representative for the user watching behavior. Now the broadcasters still work with viewing figures, which are representative for only a small group of users. The use of this data for providing better TV quality is ethically permitted.

Because everything is about money, the danger is that only programs with high viewing figures will be shown on television. But these days, the most high quality TV programs are only broadcasted by public broadcasters. So they don't depend on publicity money and they will not disappear because of this system.

The results of this method will probably turn out to be better than the results of the individual method. Because five hundred opinions will provide us with a lot more information than only 1 opinion. Because of that, the program can be evaluated more accurately.

For the service-provider, the advantage of gaining such detail viewing figures is worth mentioning. Having that advantage, the service-provider will probably ask less for providing the XML Meta data. This lower cost to join the service is an advantage for the users.

## 8.6 Functionality

This method recommends the programs the users like and disapproves the programs the users dislike. In general, this method is approved by 74% of the users. It means that 74% of the users liked the programs which were suggested. An inquiry was done to state this number as you can see in figure

8.3. The investigation was done on a randomly picked group of 200 persons. They were asked to rate 50 TV Programs. After that was done, they were send both a list of suggested TV programs and a list of disapproves TV programs. They were asked their opinion on the recommendation. They could choose between 'I like it', 'neutral' or 'I dislike it'.

## 8.7 Possible improvements

We already mentioned that program suggesting is an iterative process. At any time, a user can rate a program. Doing that, the user gets a more and more specified UserProfile. The program, knowing the users preferences will be better at suggesting new TV programs to the user.

Some times, it can be a good thing to recommend a new type of program to a user. That way the user will not come to a standstill in his television watching behavior. The program has to help the user to discover new programs.

This methods appreciation will not change a lot after some iterations. Because everybody's opinion is taken into account, we are sure that the general appreciation of a program will never be extreme. That way, this method will never give extreme recommendations and will have a very stable appreciation over time.

# Chapter 9

# The Community based Method

This last method uses **communityProfiles** for suggesting TV programs. Community-detection is the process of detecting groups of people who have the same **interests** or **characteristics**. Community-building is the process of joining these people in groups, and mapping their characteristics to the entire group. That way a **comunity-profile** can be composed to represent the properties of a community.

If we know a user belongs to a certain community, we can assume how he will react on something in accordance with the behaviour of his community. This is a powerful technology which can be used in a lot of positive (and also negative) ways. A positive example is **personalisation**. A negative example is **personalised publicity**.

So, just as in the previous method, this method uses a combination of **collaborative** and **content based filtering**. With the difference that the content based part will only take in profiles from users that are classified in the same **community** as the first user.

In this chapter we will discuss how we can recommend programs using **both** the individual profile of a user and the profiles of the people from the users' community. This should logically be the best method of them all, because of the following two principles. First it is logical that the more opinions we have, the more tinformation we have, the better the program suggestion will be. But secondly, the opinions that are taken into account, are only from people from the same community – with the same interests. Therefore, this method should logically result in better TV program suggestions.

Every user has his personal UserProfile stored locally and sends it to the server. There, the **clustering** is done to detect groups of users with the same interests. This results in a communitySet. The **CommunitySet** is then send through to all the users. So for validating a program, we can now

look at the individual profile to obtain a personal opinion on the program. But we can also look at the profiles from the users who had the same interests as the first user. That way, unknown programs may be recommended with more certainty of success.

## 9.1 The Algorithm

This method takes the meta-data from a TV-program-schedule, a userProfile and the **communitySet**. The latter is send to the user by the service-provider. It is clear that the computing of a communitySet is done on the server of the service-provider. A community is specified by a **communityProfile**. That is the union of all the profiles from the users from the community. This method will combine the individual method (described in the chapter 7) with the general opinion the other users from the same community have on a certain program. It will output a score telling the user whether this TV-program is suggested or not. At that point, the user still can choose which TV-programs he watches.

Just like in the previous methods an initial userProfile has to be created. When a user starts up the system for the first time, his opinion is asked on fifty TV-programs randomly picked out of the TV schedule. The user can rate the programs with one of the following rates: 'not known', 'very bad', 'bad', 'neutral', 'good' or 'very good'. When the user chooses 'very bad' for example, all the keywords from that program get -2 on their score in the userProfile. Similarly the keywords get adapted with scores from -2 to +2 depending of what the rate of the program was. That way, an initial userProfile is created.

How the communitySet is provided to the user is variable. The first option is that the server sends the communitySet to every user who has joined the service. That should mean that the users' set top-box should have an open connection for receiving a file at any time. The second option is that the users request the server for the communitySet-file whenever they want it. It is clear that **not every** day an update on the communitySet is required. But the period of time in which an update on the communitySet is downloaded from the server can vary depending on the business model from the service-provider. Similarly the community-detection process has to be done at specified times.

### 9.1.1 Classifying the user in a community

When a communitySet is stored on the local hard disk of the users set top-box, we can work with that communitySet until an update is requested. First we have to find the community (or communities) in which the user fits. Because the userProfile is changing continuously, the matching to the

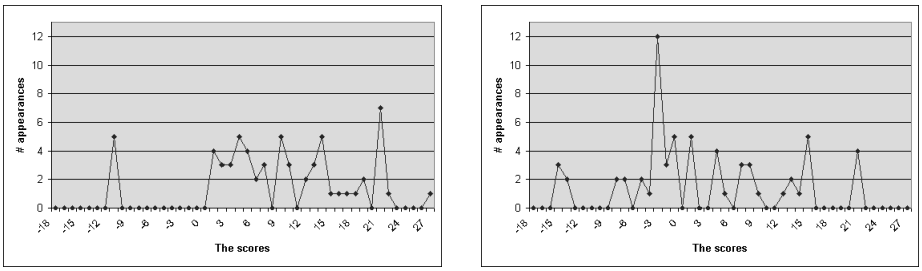


Figure 9.1: The scores distribution of 62 programs, based on the profiles of community 1 (the left) and of community 2 (the right)

communitySet has to happen every time we want to evaluate the TV programs. When that is done, we can start the TV program recommending process.

The communities are specified by a **communityProfile**. A communityProfile is much the same than a user-profile, except for the scale. The classifying method will loop over the communitySet and see whether the communityProfile matches with the userProfile. When that is the case, the user is classified in that community. Note that it is possible that a certain user fits in several communities. But because in the community-detection a communitySet is searched which has the least number of users who overlap in two communities, the majority of the users will only be classified in one community.

The formula that is used to see whether a user fits in a community or not, must be the same that is used in the community-detection process. Note that it is possible that a user is not classified in a community at all. In that case the individual recommending method should be used for that user.

### 9.1.2 Scanning the userProfile and the communityProfile(s)

Similarly then in the previous methods, a scan is made in the userProfile to see what the average score of the persons is. If the general opinion on the TV programs is higher than 0, we have to take that into account suggesting a program or not. This scan results in the *user-threshold*. The same is then repeated in the communityProfile from the user's community, which called the procedure. This scan results in the *Community-threshold*. This has to be done for every community the user is in. All the thresholds are stored and propagated through to the rest of the method.

#### Why?

The reason for obtaining the user-threshold came clear in a previous chapter. If you want to know more on that I would refer to section 7.1.1.

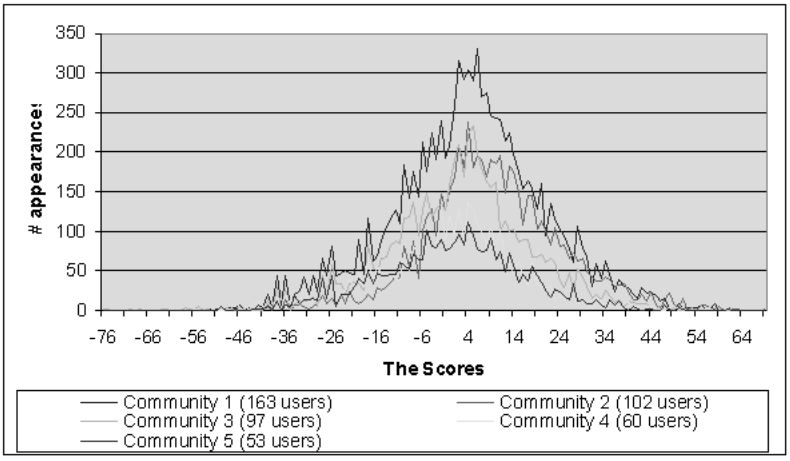


Figure 9.2: The scores distribution of 62 programs, given by the users grouped by communities

The reason we want the Community-threshold is in fact the same as the reason of why we wanted the user-threshold. We want to know with what kind of people we are dealing with. It can possibly be that the general opinion on the programs is very high. This could mean two things: The users are pleased with the television schedule or the users easily give good rates. Because we depart from the vision that the users **only** wants *good* programs suggested. The user will prefer less suggested programs, but all good ones above the suggestion of a lot of programs where some bad are included. Wanting to suggest only the relative best programs, we have to take into account how the general opinion is. If it is 'good', we should be more critical suggesting a program, than if it were 'neutral'.

In figure 9.1 we observe the different valuation of 62 programs by 2 communities. Here we see that the first community on average gives higher scores to the programs, while the second community gives lower rates. This can mean that the TV programs offered were more suited for the first community. But when we are now suggesting programs to people from the first community, we can be more critical without having to decline every TV program. For the second community, we won't have to be a lot more critical, because otherwise, all the TV programs would be declined. But the opposite won't be good either, because than we would start to suggest programs that the users don't like. And that's not what we want at all. We **only** want **good** programs to be recommended.

**How?**

For calculating the community-score of a TV program, we are working with the communityProfile. So we have to divide the end-result by the amount of users from that community. This results in average scores for the TV programs. At the end we have a score-partition which is always less extreme than with individual users (like you can see in figure 9.1. But certain types of programs will come out with this method. In the same figure you can also see the different valuation of two different communities. In this example, we see that the first community has a higher average score then the second community.

As we see in figure 9.1, the scores a program can get in this example, are always situated between -18 and +27. This is not as extreme as the individual scores like you could see in figure 9.2. In that figure, the individual scores of the programs are shown. As we can see, the program scores here are a lot more extreme fluctuating between -76 and +69. If the communities consist of enough users, we see that the score distribution for each community follows a **Gaussian distribution**. After having done this experiment several times with different communities, we notice that the scores distribution of one community **always** follows a Gaussian distribution.

In reality we won't have the different users opinions on the programs locally on the set top-box. So three possibilities remain for obtaining the community-threshold. The first is to send through the UserProfileSet from the server to the set top-boxes and calculate it like in the previous chapter. But that would mean a lot of transmitting and so less scalablility. The second option is to do the necessary calculations on the server. Then the communityProfile and some calculated scores must be transmitted (which scores will be discussed below). The last solution would be to work just as we did in the first method. In this solution we would treat the communityProfile just like the user-profile. We would work with a linear partition for finding the adequate boundaries.

Knowing that the score distribution always follows a Gaussian distribution, we know that working with a linear partition doesn't give correct results. So that would eliminate the third option. Knowing we have more calculating power on the server and taking into account the transmission overhead the first method would create, the second option seems to come out the best. So the calculations will be done on the server side and the results will be send, together with the communityProfiles to the clients. So the method of the previous chapter can be used to calculate the community-thresholds $ct_t$. The only difference will be that we have to execute this procedure for each community.

For obtaining the average validation of the programs, we now know we have to calculate the $E(X)$ of each community. Just like in the previous method, the principle will be used within each community:

X is the validationScores a program gets from a user from community $C_t$ with a userProfile UP. The number of validationScores is $p$, then

$$ct_t = E(X)_C = \sum_{i=1}^{p}(x_i.P(X = x_i))$$

is the expectation value for X from a certain user. As you can see, we need to know the probability of $X = x_i$. This can be calculated by solving the following equitation:

$$P(X = x_i) = \frac{\#\text{ of measures where } X = x_i}{\#\text{ of measures done}}$$

The number of measures done, is equal to $\#Programs.\#UserProfiles_C$. So we only have to know the number of measurements where X equals $x_i$. That amount we can retrieve in the NumberVector on index $x_i$, just like in the previous method.

In the example of the graph 9.2, the values of E(X) differ from community to community. With this number, we can also know whether a specific community 'liked' the overall TV-schedule. The value of E(X) will be outputted as the userSet-threshold.

### 9.1.3 Rating the TV programs

After obtaining the Community-thresholds, the results (the values of the $ut_C$'s of all the communities together with the communityProfiles have to be provided to the user. Then the local calculations can take place to compose a personal TV schedule. The first thing we have to do is see in which communities the user fits as discussed in 9.1.1. The procedure then calculates the personal score by looping over the keywords of the program meta-data. For each keyword, it looks up the score in the userProfile of the user and adds it to the PersonalScore. After that is done, it loops over the communityProfiles of the communities where the user is in. Then the average scores the community-people would give to that TV program are calculated. Similarly to the previous chapter, we define:

$$\begin{array}{rclrcl} P & = & \{p_i = program \in Televisionschedule\} & |P| & = & p \\ K_i & = & \{k_j = keyword \in p_i.MetaData\} & |K_i| & = & k \\ UP & = & \{k_up = keyword \in UserProfile\} & |UP| & = & n \\ SET & = & \{C_t = communitytheuserisin\} & |SET| & = & h \end{array}$$

The principle, which is used to calculate the Community-scores for a TV program, is the same as the one we used to calculate the PersonalScore. This is all summarized in the following pseudo-code:

$\forall p_i \in P$ {

```
int PersonalScore = 0;
```

```
int ComScore = 0;
int TempScore = 0;
∀k_j ∈ K_i {
   int PersonalScore = PersonalScore + PUP.getScore(k_j);
}
∀C_t ∈ SET {
   ∀k_j ∈ K_i {
      TempScore = TempScore + C_t.getScore(k_j);
   }
   TempScore = TempScore/C_t.getNumberOfUsers();
   ComScore = ComScore + TempScore;
}
ComScore = ComScore/h
Score(p_i) = ((PersonalScore − ut) + (ComScore − ct_t))/2
}
```

Where $PUP$ is a userProfile of a certain user with user-threshold $ut$ and $\#K = k$. $p$ is the number of the programs rated by the user. $h$ is the number of communities in which the user is classified. $C_t$ is a Community with CommunityThreshold $ct_t$.

$$Score(p_i) = \frac{\sum_{j=1}^{k}(PUP.getScore(k_j)) - ut}{2} + \frac{\frac{\sum_{t=1}^{h}(\sum_{j=1}^{k}(C_t.getScore(k_j)))}{h} - ust}{2}$$

OR

$$Score(p_i) = \frac{\sum_{j=1}^{k}(PUP.getScore(k_j)) - \frac{\sum_{i=1}^{p}(x_i))}{p}}{2} + \frac{\frac{\sum_{t=1}^{h}(\sum_{j=1}^{k}(C_t.getScore(k_j)))}{h} - \sum_{i=1}^{p}(x_i.P(X = x_i))}{2}$$

is the score the program gets from that user.

### 9.1.4 Suggesting a TV program

Now the question rises how we know when a program $p_i$ with its score Score($p_i$) is classified under a certain category. A program can be classified in one of the following five categories: 'very bad', 'bad', 'neutral', 'good' or 'very good'.

| **Rate** | **Percentage** | $\frac{Score(P)-\mu}{\sigma}$ |
|---|---|---|
| 'very good' | 10% of the programs | 1.28 |
| 'good' | 20% of the programs | 0.53 |

Table 9.1: Normal Distribution Function

**The Gauss distribution**

We know that the graph of the X = score($p_i$) is always following a Gaussian distribution $N(\mu, \sigma)$. When we now know that we want only 10% of the TV programs to be classified in the 'very good' category, we can calculate easily that:

$$\begin{aligned} P\Big(X \leq \frac{Score(p_i)_{\text{very good}} - \mu}{\sigma}\Big) &= 10\% = 0.1 \\ &\Updownarrow \text{ using the statistical tables} \\ 1.28 &= \frac{Score(p_i)_{\text{very good}} - \mu}{\sigma} \\ &\Updownarrow \\ Score(p_i)_{\text{very good}} &= (1.28.\sigma) + \mu \end{aligned}$$

**The Classification**

Similarly the following borders can be calculated for the good and 'neutral' rates. For the 'bad and 'very bad' rate, we can make use of the symmetrical property of the Gaussian distribution. Knowing that $E(X) \approx 3$, it is easy to calculate that for being categorized in the 'very bad' category, a program needs to have a score of:

$$\begin{aligned} Score(p_i)_{\text{very bad}} &= E(X) - \Big(Score(P)_{\text{very good}} - E(X)\Big) \\ &\Updownarrow \\ Score(p_i)_{\text{very bad}} &= E(X) - \Big((1.28.\sigma + \mu) - E(X)\Big) \\ &\Updownarrow \\ Score(p_i)_{\text{very bad}} &= E(X) - 1.28.\sigma - \mu + E(X)) \\ &\Updownarrow \\ Score(p_i)_{\text{very bad}} &= 2.E(X) - 1.28.\sigma - \mu \end{aligned}$$

On overview of the results of the calculations is given in table 9.2.

| Rate | Percentage | Interval |
|---|---|---|
| 'very bad ' | 10% | $[m,2.E(X) - 1.28.\sigma - \mu]$ |
| 'bad' | 20% | $[2.E(X) - 1.28.\sigma - \mu,2.E(X) - 0.53.\sigma - \mu]$ |
| 'neutral' | 40% | $[2.E(X) - 0.53.\sigma - \mu,0.53.\sigma + \mu$ |
| 'good' | 20% | $[0.53.\sigma + \mu,1.28.\sigma + \mu]$ |
| 'very good' | 10% | $[1.28.\sigma + \mu,M]$ |

Table 9.2: Program Scores Classification

## 9.2 Performance

In this method, five things are in fact happening:

- Classifying the user in a community
- Scanning the UserProfile
- Scanning the CommunityProfiles
- Calculating the classification borders
- Looping over the TV programs and giving scores

### 9.2.1 Classifying the user in a community

The code that will be executed here is dependant of the code that was usedfor doing the community-detection on the server. Therefore we don't know exactly what the performance of this procedure will be. But we do kow an upper bound for the performance. The procedure will certainly have to take in a communitySet and the personal userProfile. Then a loop will have to be started to compare the userProfile with the communtyProfile. If enough similarities are found, the user will be classified in that community. But, becusethe possibility remains that a user is classified in more than one community, we have to continue the loop. This can all be summarized in the following pseudo-code:

$\forall C_t \in C_t$ in communitySet {
  $\forall K_i \in UP$ {
    $C_t.getScore(K)$
  }
  `// do the neccessary calculations with the data`
}

Now, we can make the calculations for the worst case scenario. This will give an estimation on how long it will possibly take to decide in which communities a user fits.

$$
\begin{aligned}
T(|communitySet|, k, n) &= |communitySet| .k. \log(n) \\
\rightarrow O(ClassifyUsers) &= O\Big( |communitySet| .k. \log(n) \Big)
\end{aligned}
$$

### 9.2.2 Scanning the userProfile

Just like in the previous methods, the performance of scanning the userProfile is totally independent of the amount of users that joined the service. Because the scanning only needs the personal profile, it can execute locally on the set-top box of the client. It only depends on how detailed the profile is (how much keywords are stored in the profile). Earlier in this chapter, we defined this number as $n$.

The same pseudo-code is executed as in the previous chapters for obtaining the user-treshold. Therefore it is easy to see that the same execution time will be needed here.

$$
\begin{aligned}
T(p, k, n) &= p.k. \log(n) \\
\rightarrow O(ScanUserProfile) &= O\Big(p.k. \log(n)\Big)
\end{aligned}
$$

Just like in the previous chapter, we see that it consists of two parts; an exponential and a logarithmic part. But because the service provider can decide the values of $p$, $k$ and $n$, it is not a problem. The advantages off incrementing them should be compared with the loss of performance in terms of execution speed.

### 9.2.3 Scanning the CommunityProfiles

As you could see in section 9.1.2, the scanning of the CommunityProfiles is a bit more complex. This results in a longer computation time. If $p$ is the number of programs looped on, $k$ the number of Keywords in the meta-data of one TV program. Let $n$ be the number of Keywords in the userProfile and $h$ the number of Communities that the user is in.

$$
\begin{aligned}
T(h, p, k, n) &= h.p.k. \log(n) \\
\rightarrow O(ScanCommunitySet) &= O\Big(h.p.k. \log(n)\Big)
\end{aligned}
$$

But remembering this will be done on the server of the service-provider, it doesn't matter for the clients and will therefore not be counted in the final conclusion. Instead $h$ will be counted because we will still have to retrieve the correct community-threshold in a vector.

### 9.2.4 Calculating the classification borders

Just like in the previous methods, the calculation of the borders is independent of the amount of users that joined the personalization service. Its execution happens locally on the client's set-top box. For calculating the borders we need the statistical value that can be found in the table of Gauss (table 9.1 shows an extract of the needed values). But because we only need the values for 10% and 30% we don't need the entire table. Also the values of E(X) and V(X) are needed to make the necessary calculations.

$$1.28 \quad = \quad \frac{Score(p_i) - E(X)}{V(X)^2}$$

The values of E(X) and V(X) can be calculated knowing the values of $\sigma$ and $\mu$ of the communities Gauss Distribution. Those values were calculated on the server in the scanning of the CommunityProfiles. Then they were sent through together with the communitySet. We know that:

$$\begin{aligned} \sigma^2 &= V(X) \\ \mu &= E(X) \end{aligned}$$

The exact calculation of $E(X)$ and $V(X)$ is not complicated knowing the values of $\sigma$ and $\mu$. So the seven calculations that have to be done in this step, are very small and will go very fast.

### 9.2.5 Looping over the TV programs and giving scores

This procedure will loop over all the TV programs, which are offered in the XML schedule. A score has to be calculated for each program, telling the user whether this program is suggested or not. The way the score of a program $p_i$ is calculated, can be seen in section 8.1.3. We see that it consists of two parts: calculating the PersonalScore and calculating the ComScore.

Calculating the Personal score of a program is done simultaneously with the scanning of the UserProfile. So the only thing what remains is the retrieving of the adequate personal score of a TV program, which is of $O(log(p))$ where p is the amount of TV programs in the XML schedule. It is logarithmic because a binary search method is used to retrieve the correct title.

The performance of the second part depends on four things: the order of the getScore-method, the number of Communities the user is in, the amount of keywords characterizing each program $p_i$ and the number of programs to be processed. Knowing the amount of keywords characterizing a program is $k$, the amount of programs included in the XML-schedule is $p$, the number of

Communities the user is in is $h$ and that getScore is of the order $O(\log(n))$, the second part is of order:

$$\begin{aligned} T(h,p,k,n) &= \log(p) + h.p.k.\log(n) \\ \rightarrow O(Looping) &= O\Big(\log(p) + h.p.k.\log(n)\Big) \\ &= O\Big(h.p.k.\log(n)\Big) \end{aligned}$$

### 9.2.6 Conclusion

The total performance of this first method is the sum of the order

$$\begin{aligned} T(m,p,k,n) = & \underbrace{|communitySet|\,.k.\log(n)}_{6.2.1} + \underbrace{p.k.\log(n)}_{6.2.2} \\ & + \underbrace{h}_{6.2.3} + \underbrace{7}_{6.2.4} + \underbrace{\log(p) + h.p.k.\log(n)}_{6.2.5} \end{aligned}$$

$$\begin{aligned} \rightarrow O(ProgramSuggestion) &= O\Big(|communitySet|\,.k.\log(n) + p.k.\log(n) \\ &\quad + h + 7 + \log(p) + h.p.k.\log(n)\Big) \\ &= O\Big(h.p.k.\log(n)\Big) \end{aligned}$$

Knowing that the service provider can choose the maximum of $h$, $p$, $k$ and $n$, we can see that this method for suggesting TV programs still has a good performance. This is mainly caused by making the server do the biggest calculations and sending through the results. The update on these results, will happen together with an update on the CommunitySet.

## 9.3 Architecture

At client side no big difference will be detected comparing with the previous methods. Compared with the first method, there are only two new elements on the client side. The first is an addition of a module that can cope with the receiving of a CommunitySet. The second is a module that can send the personal userProfile to the Server. As we see here, we do need an uplink to the server to use this method. Therefore only the **remote storage** is considered here. We could also use a **local storage** architecture, but then the necessary changes must be made to obtain an uplink to the server.

On the server side, things get more complex now. Because now the amount of data send to the server is a lot bigger than in the first method,

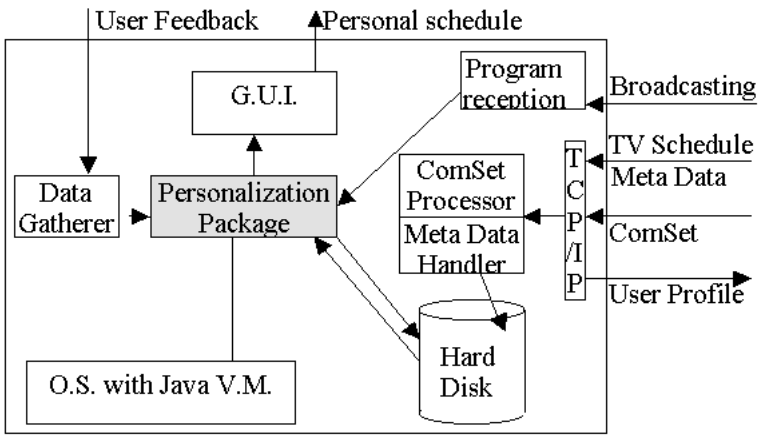


Figure 9.3: The Community based method architecture

we have to take hard- and software precautions. The architecture of the clients set-top box is shown in figure 9.3.

## 9.4 Data and data transfer

On the Hard Disk of the set top-box, the entire TV-schedule for one week is stored in XML format. Every day's television schedule has a separate file identified with an exclusive number. That number identifies the file. When a day is done, the XML file that contained the television schedule of that day can be removed from disk. Also the UserProfileSet has to be stored locally on the hard disk.

There are three types of data transfer that have to take place in this method. The first is the acquisition of new TV-schedules by the client. The second is the acquisition of the CommunitySet together with the values of $\sigma$ and $\mu$ for each community. The third is the sending of the personal userProfile to the server, which collects all the userProfiles for doing the community-detection process. These transfers can be done day by day. Every morning, for example, the set top-box could make a connection with the server, requesting an update on the television schedule for the coming week and on the CommunitySet. At that time, the set top-box compares the file-ID's it has on its hard disk with the file-ID's on the server. If the files-ID's that overlap remain the same, and the files with new ID's are downloaded from the server to replace the older files. The CommuniySet with the values of $\sigma$ and $\mu$ for each community has to be replaced entirely.

If a change to the television schedule is made on a certain day, the server's file of that day will be renamed. When the client requests the server for an update on the television schedule, the set top-box will notice the change,

and it will update the file as explained above.

For sending the personal userProfiles to the server, the server needs an architecture that is *scalable*. That means the server must be able to cope with a lot of client-connections at the same time.

## 9.5 Ethical aspects

This method may cause the same ethical problems that are stated in the previous 2 chapters. But in this method, another ethical aspect rises. The detection of groups of users - communities - and the classifying of a user in a certain community, are not correct from an ethical point of view.

In this method, we create a community with a communityProfile. That profile serves as the profile of a stereotypical person. That person will be the model user for the entire community. This way, we assume that every person in the community is the same and can be represented by the stereotypical profile. But who are we to think those persons all think alike?

It is OK to group the users into communities, but it should be better to take into account every persons **individual** opinion, for valuating a TV program. That way, every person is seen as a separate entity.

The classic broadcasters still rely on community-building based on qualitative data. A classic example is the broadcasting of children's programs around 5 PM. In this example, they target a community that is composed of children who come home from school at that time. So, for those broadcasters it depends how old you are to be classified in a certain community. That is ethically not permitted.

Therefore, we should be well aware that in our system only the quantitative data is used to classify the users in communities. Your gender should not have anything to do with your TV program preferences! That is assured in this method, because we only gathered quantitative data in the first place.

## 9.6 Functionality

This method recommends the programs the users like and disapproves the programs the users dislike. In general, this method is approved by 65% of the users. It means that 65% of the users liked the programs, which were suggested. An inquiry was done to state this number as you can see in figure 9.4. The investigation was done on a randomly picked group of 200 persons. They were asked to rate 50 TV Programs. After that was done, they were sent both a list of suggested TV programs and a list of disapproves TV programs. They were asked their opinion on the recommendation. They could choose between 'I like it', 'neutral' or 'I dislike it'. This is a hope giving mark, but it is not good enough to start program suggesting.

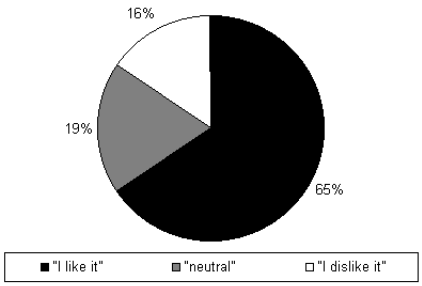


Figure 9.4: The appreciation of method 2, by 200 users

## 9.7 Possible improvements

We already mentioned that program suggestion is an iterative process. At any time, a user can rate a program. Doing that, the user gets a more and more specified userProfile. The program, knowing the users preferences will be better at suggesting new TV programs to the user. Some times, it can be a good thing to recommend a new type of program to a user. That way the user will not come to a standstill in his television watching behavior. The suggesting program has to help the user to discover new programs.

This methods appreciation will change a lot after some iterations. Because more and more specified communities will start to exist and the users will be classified in those specified communities. We expect the appreciation of this method to increment over time.

# Chapter 10

# The Inquiry

There are several reasons why an enquiry on this new technology has to be held. The first is to obtain the users opinion on the program suggestions application. The question is how many people will be interested in an application that would provide you with a personalized TV schedule. The second thing we want to know is how good the program suggestions would be with a limited XML meta-data-description. How detailed must this meta-data be to provide a good personalized TV schedule?

Another important aspect is the comparison between the different methods. The different methods were discussed in the previous chapters: **the individual method**, **the collaborative method**, and **the community based method**. Which method provides the best TV schedule?

## 10.1 Description

For obtaining answers on the questions that we asked ourselves, an inquiry was held over the Internet. Several mails were sent over the internet to an arbitrary group of people living in Belgium, inviting them to fill out a form. It was ensured that only Belgians participated, because the TV programs that had to be rated were Belgian. Sending e-mails only to addresses terminating with '.be' did this. The inquiry consists of three phases.

For the first phase a website was constructed containing a fill-out-form. In this form the users were asked to give their opinion on fifty different TV programs. For rating the TV programs, they had the choice of: 'not known', 'very bad', 'bad', 'neutral', 'good or 'very good'. When that was done, the users were asked to input their e-mail-address so that we could contact them for continuing the inquiry. At the end the form was posted to me.

Also an XML file was composed containing the meta-data on the different TV programs. As mentioned before, every TV program was only classified by four keywords. This because a real XML meta-data file on Belgian TV programs was not available for use. The TV programs from a foreign file

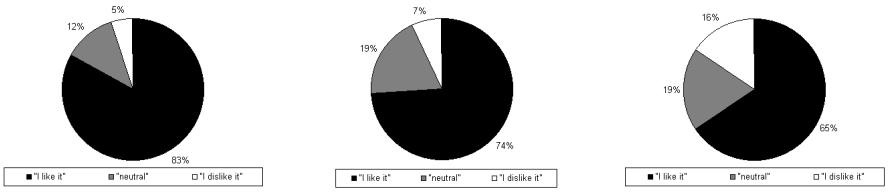


Figure 10.1: The appreciation of the methods, by 200 users (left: Method1, center: Method2, right: Method3

are not all known in Belgium. So the use of a foreign file was not a solution either.

The second phase consisted of computing the TV program suggestions for the users. On my computer a parser parsed the incoming forms. It took in a fill-out-form of a user and the XML meta-data file and outputted a User Profile. This was done for all the people who filled out the form on the Internet. After that was done, the three discussed methods were used to provide the users with a list of recommended programs. Then a mail was send to the users containing the three lists with both the suggested and the turned down TV programs. Every TV program was given a score between one and five for rating the programs too. It is clear that the turned down programs all had a score of three or lower. In that mail, the users were asked which method was the best in providing them with 'good' TV programs.

In the third phase the feedback on the different suggesting methods was received through e-mail. The users provided me with the order in which they appreciated the suggesting methods. Another parser parsed those results, and ended up with some interesting results, which will be discussed in the next section.

## 10.2 Results

On the web site where the fill-out-form was placed we counted the number of hits. The total number of requests on the page was 407. The number of forms I received was 367. That means that more then 90% of the people who read the description of the application wanted to collaborate in order to receive a personalized TV schedule. Knowing the fact that the number of TV stations is increasing, we see that the need for such a service only will get bigger. These facts indicate the need for the application.

### 10.2.1 Qualitative Results

Generally, the users approved all the methods, as you can see in the figure 10.1. But for building up a relation of trust, we need an approval of 90%

| | Method 1 | Method 2 | Method 3 |
|---|---|---|---|
| First place | 62 | 32 | 32 |
| Second place | 30 | 35 | 61 |
| Third place | 34 | 59 | 33 |
| Total Score | 280 | 225 | 251 |

Table 10.1: The evaluation of 126 users of the different methods

of the suggestions made. In these figures we see that the third method has the worst appreciation. This is logical because after just one iteration-cycle, we cannot have enough information for making a good classification of the users in communities.

The part of the users that didn't think that the recommendations were good, is still a large group of users. This can be explained by the meta-data. In the meta-data we worked with in this experiment, there were only four keywords characterizing the programs. If this number would be increased, the results would get better every time.

### 10.2.2 Quantitative Results

The users were also asked which of the methods they preferred. How the methods were evaluated is stated in table 10.1. The total score of a method is calculated as follows: three points for a first place, two points for a second place and only one point for a third place. As we can see in this table, the first method was chosen to be the best, followed by the third method. Logically the second method turned out to be the worst.

The valuation differences between the methods are not very big. This can be explained on the number of users who completed the third phase of the inquiry. Only 126 users provided feedback on the preferences they had on the suggestion methods.

We should not only look at the first iteration. Because in this experiment only one iteration was done, we only know the results after one iteration. But when some cycles are done, we expect the third method to become the best.

## 10.3 Conclusions

In this experiment, we learned that the meta-data is the heart of the entire system. if the meta-data is bad, the results will be bad too. We now also know that only four keywords for characterizing a TV program is not enough. In future experiments this parameter should be changed to compare the results with differnt amounts of keywords. That is the only way we can obtain the best value of $k$. A lot is changing on the composing of meta-data,

and only one universal standard will probably survive. Than, that standard will be used to compose the meta-data.

It turned out that making the difference between some TV programs does not only depend on the genres the program is classified in. But also the feelings the TV program causes with the viewers. So maybe those feelings should also be included in the meta-data.

We learned that a combination of the first and the last method would be the best for making a program suggesting application. Because in the beginning the User Profile is not specified enough, we should start suggesting using the first method. After a couple of cycles, the third method should take over control. This should result in the best TV program recommendations.

# Chapter 11

# Conclusions

We started this thesis with a profound literature study on the topic Personalization. Then we specialized more in Personalization for the broadcasting industry, where Digital TV is coming up fast. Knowing that Digital TV will allow Personalization in the TV industry, the idea was formed to make a TV program suggesting system.

This thesis shows an example of each step that has to be taken to build such an application. First there was the data gathering. This is the process that collects the data on the users for building profiles. Then there was the community-building phase. In that phase the similarities between the user profiles are detected to form groups of users with the same characteristics.

Finally, there is the program-suggesting phase. This is the phase where the emphasis lies on in this thesis. We investigated and compared the different TV program suggesting methods: the individual method, the collaborative method and the community-based method. An inquiry was held to obtain the users opinion on those methods.

## 11.1 Conclusion

Personal TV completely changes one's interaction with television. It turns television into a tool for furthering ones education or, depending on your preferences, greatly increasing the entertainment value. Personal TV certainly has the potential to greatly concentrate the quality of the content consumed.

It is clear that Personal TV will become the standard means of interacting with television content. Our world simply does not have the patience to tolerate a push-only media with such a high concentration of content that is completely uninteresting.

Though, it turned out that such a system is highly dependant of metadata. Metadata management is the key to the success of the Personalization technology, and places a burden on broadcasters, as it has to be accurate,

complete and consistent. If multiple standards and platforms are to be supported, the data overhead becomes large and expensive. So one could hope a universal standard to come out in the near future. In the inquiry, only four keywords were used to characterize a TV program. That number turned out to be too low.

Personalization in this domain comes in two different architectures: local storage and remote storage. While the first stores the data of a TV program locally on a hard disk in the set-top box, the latter will only send the requested programs to the users. Whereas the individual method can be used in both the architectures, the collaborative and community based methods can only be used in a remote storage architecture, unless the necessary adaptations are made.

For making the best TV program suggestions, we should use a combination of the individual method and the community-method. As we could see in the inquiry-results, the individual method gave the best results in the first iteration. But after some iterations, the community-based method gets better appreciation. So a method that combines both will give the best results. It should start using the individual method and after a while switch over to the community-based method.

Ethically, some problems may rise in the personalization service. We have to make sure we keep our goal in mind. That goal is to provide a service to the users – to provide them with personalized television schedules. We may not misuse the personal data of the users for other purposes.

But if we compare this system of television with the one that is used now, we see that ethically a step foreward is made. Now, the broadcasters treat the users as a group of people with one stereotypical profile. That approach has two bad consequences. The first is a result of the kind of profiles they use. For them, it matters what gender you have and how much money you earn. In the new system, the userProfile will no longer contain qualitative information but only quantitative information. The second drawback is that individual opinions do not count. In the new system, every users personal preferences are taken into account to provide a personalized television schedule.

## 11.2 Future Work

It is clear that the resources with which I could work, were very low. For example I had to compose my own meta-data on the TV schedule. If some real Meta data could be provided to work with, the appreciation results of the methods would certainly turn out to be higher.

The user set on which the inquiry was done, should be bigger too. I could only work on a group of less than 400 users. The inquiry was done in 2 cycles. In the first cycle those 400 users participated, but on the second

cycle, only 140 users participated. A third cycle could not be held, because the number of users was not representative anymore.

The best way to obtain correct results is to do the inquiry in a real life environment. Because the system is already used by a large number of users in the USA and the UK, this can be done in one of those two countries. Also the real execution speeds can be measured from the moment we are really running the system on a set top-box.

# Bibliography


[1] O. De Troyer: *User Aspects of Software systems*, VUB course slides, 1999.

[2] Refsnes Data: *W3schools.com*
http://www.w3schools.com/

[3] N. Wells, J. Wolfers: *Finance with a personal touch*, Communications of the ACM, 2000.

[4] U. Manber, A. Pathel, J. Robinson: *Personalisation on Yahoo*, Communications of the ACM, 2000.

[5] D. Riecken, K. Early: *Personalised Communication Networks*, Communications of the ACM, 2000.

[6] J. Kramer, S. Noronha, J. Vergo: *A User-centered design approach to Personalisation*, Communications of the ACM, 2000.

[7] J. Karat, C.M. Karat, J. Ukelson: *Affordances, Motivations and the Design of User Interfaces*, Communications of the ACM, 2000.

[8] E. Schonberg, T. Cofino, R. Hoch, M. Podlaseck, S.L. Spraragen: *Measuring Succes*, Communications of the ACM, 2000.

[9] N.J. Belkin: *Helping eople find what they don't know*, Communications of the ACM, 2000.

[10] L. Candy, E. Edmonds: *Creativity Enhancement*, Communications of the ACM, 2000.

[11] M. Minsky: *Commonsense-based Interfaces*, Communications of the ACM, 2000.

[12] J. McCarthy: *Phenomenal Data Mining*, Communications of the ACM, 2000.

[13] E.P.D. Pednault, E. Edmonds: *Representation is Everything*, Communications of the ACM, 2000.

[14] E. Volokh: *Personalisation and Privacy*, Communications of the ACM, 2000.

[15] D. Riecken: *Personal End-User Tools*, Communications of the ACM, 2000.

[16] D.C. Smith: *Building Personal Tools by Programming*, Communications of the ACM, 2000.

[17] P. Maglio, R. Barret: *Information Streams*, Communications of the ACM, 2000.

[18] H. Hirsh, C. Basu, B.D. Davidson: *Learning to Personalize*, Communications of the ACM, 2000.

[19] B. Smyth, P. Cotter: *A Personalized Television Listening Service*, Communications of the ACM, 2000.

[20] P.B. Kantor, E. Boros, B. Mejamed, V. Menkov, B. Shapira, D.J. Neu: *Capturing human Intelligence in the Net*, Communications of the ACM, 2000.

[21] W. Zadrozny, M. Budizikowska, J. Chai, N. Kambathala, S. Levesque, N. Nicolov: *Natural Language dialogue for Personalised Interaction*, Communications of the ACM, 2000.

[22] M. Percowitz, O. Etzioni: *Personalising Web Sites*, Communications of the ACM, 2000.

[23] T. Nakayama, H. Kato, Y. Yamane: *Discovering the gap between Web Site Designers'expectations and Users'behavior*, Communications of the ACM, 2000.

[24] Sun Microsystems, Inc: *The Java API*, 1999.

[25] E.J. Friedman-Hill and Sandia Corporation: *The JESS API*, 1999.

[26] D. Flanagan: *Java In a Nutshell*, O'Reilly 1997.

[27] E.J. Friedman: *Jess, The Java Expert System Shell*, 1997.
http://herzberg.ca.sandia.gov/jess/

[28] B. Marchal: *Introduction to SGML*, 2000.

[29] Willcam Group: *Willcam's Comprehensive HTML Cross Reference*, 1998.
http://www.willcam.com/cmat/html/crossref.html

[30] World Wide Web Consortium: *W3C Standards, W3C news, W3C technologies*, 2000.

http://www.w3.org/

[31] M. Anthony: *CSS tutorial*, 2000.

http://www.dynamicdeezign.com/css/introduction.html

[32] A. SiliconValley: *What is XML?*, 1999.

http://www.geocities.com/SiliconValley/Peaks/5957/xml.html

[33] J. Bosak: *XML, Java, and the future of the Web*, Sun Microsystems, 1997.

[34] H.S. Thompson: *An Introduction to XSL*, HCRC Language Technology Group University of Edinburgh, 1997.

[35] S. Salvo: *An Introduction to the DOM*, S.tygian B.lacksmith S.tudios, 1998.

http://zero.zero.xs2.net/studio/tutorials/salvo07.html

[36] R. Cover: *Extensible Stylesheet Language*, The XML Cover Pages, 2001.

http://www.oasis-open.org/cover/xsl.html

[37] J.J. Kuslich: *Introduction to JavaServer Pages*, 1999.

[38] J.J. Kuslich: *Introduction to JavaServer Pages*, 1999.

http://tis.eh.doe.gov/ohre/roadmap/achre/chap12.html

[39] The Apache Software Foundation : *Apache Software Foundation website*, 2000.

http://www.apache.org/

[40] Caucho: *Caucho site*, 2001.

http://www.caucho.com/

[41] A. Jo Kim: *Community building on the web*, Amazon, 1999.

[42] Yahoo!: *The Friends Community*, Yahoo!, 2001.

http://ytv.yahoo.com/fc/ytv/friends

[43] C. Forrester: *The Business of Digital Television in 2000*, Butterworth-Heinemann, 2000.

[44] M. Harries: *Report on Personal Television*, MediaGeniX, 2000.

[45] C. Zorff *Everyone wants to make a settop box*, Net4TV Voice, 1998.
http://net4tv.com/voice/Story.cfm?storyID=1188

[46] A. Chislenko *Automated Collaborative Filtering*, 1997.
http://www.lucifer.com/ sasha/articles/ACF.html

[47] E. Castro *HTML 4 for the World Wide Web Visual Quickstart Guide*, Peachpit Press, 2001.

[48] P. Mikros *Standards of Meta Data*,European broadcasting Union, 2000.
http://www.ebu.ch/tech_info.html