Client Application for KBN

(Personal IPTV Programme Guide)

Zhou Xu

A dissertation submitted to the University of Dublin, Trinity College

in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science

September 2007

DECLARATION

I declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Name: Zhou Xu

Date: 14th September, 2007

PERMISSION TO LEND AND/OR COPY

I agree that Trinity College Library may lend or copy this dissertation upon request.

_____r_____

Name: Zhou Xu

Date: 14th September, 2007

ACKNOWLEDGEMENTS

Many thanks to my supervisor, Dr. David Lewis, for the considerable time spent assisting me on this project, and for all the valuable advice and guidance offered. Likewise, I would also like to thank Dr. John Keeney and Mr. Dominic Jones, for all their work and helpful suggestions.

To all my family and friends, and especially my classmate, Yu Cao, I would like to thank you for all your support and encouragement throughout the course of this dissertation.

Zhou Xu

University of Dublin, Trinity College

September 2007

ABSTRACT

A knowledge based network (KBN) is a content based network with utilising ontology comparison and bag operations. Compared with traditional centralized client-server web applications, KBN's adopt the Publish-Subscribe model for communication between loosely coupled producers and consumers.

Rather than using full message types, KBNs match the contents of messages from producers and the attribute description from subscribers so that the message will only be delivered to interested parties. The above features, loosed-coupling and event filtering, allow for great scalability in highly distributed systems. Based on Dominik Roblek's work in his M.Sc in Computer Science (Networks and Distributed Systems) project last year, ontology comparison and a bag comparator matching algorithm have been implemented to support semantic distribution within the KBN. This project presents an implementation of a Personal IPTV Programme Guide, which applies ontology comparison and the bag comparator matching algorithm to a realistic client application for the KBN. Because it is the first time for KBN to face public users, this project focuses on the usability of such a client application. The design emphases the balance between the KBN required knowledge level and user knowledge background, including ontology concepts and using of operators. A usability test is also provided.

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

Chapter 1 Introduction

This chapter introduces the motivation of this dissertation and the process of building the project. It is followed by the objectives planed to be achieved, and concludes with a summary of the dissertation roadmap.

1.1 Motivation

1.1.1 Project Motivation

The primary motivation of this project is to extend KDEG's knowledge based network system, which has been developed by Dominik Roblek in his NDS project last year [3]. The potential and benefits of the KBN to support highly distributed autonomic system have already been discussed and evaluated [1]. But all these are were conducted in a lab by experts and researchers in this field, the KBN has not been applied in to any realistic application and faced to the end users. This project aims to apply the KBN, especially its support for ontological comparisons, into some realistic client application so that the usability could be tested and evaluated from the user experience.

1.1.2 Potential Application

Before designing and implementing the project, it is necessary to figure out what role the KBN should play and in what kind of application.

As a network, there are two kinds of inputs and one output for KBN. The input includes Notification and Subscription. A Notification could be described as a set of attributes with values. And a Subscription is also a set of attributes, but followed by operators and values to describe the desired interests. Use of a Subscription as a filter, and compare each corresponding attribute (field) with a Notification using logical conjunction, if the result is true, we can call it a match. The responsibility of the KBN is to find out all the matches between Notifications and Subscriptions, then deliver the match, the output of KBN, to

the interested party who has submitted the corresponding Subscription.

Based on this content-based event delivery mechanism, the potential client application for KBN could have the following features:

1) Pub/Sub system

The KBN is an enhanced content-based network, which loosely couples the relationship between the producer and subscriber. The delivery of a message depends on whether the message content matches the constraints described by a subscriber. Based on this feature, KBN is suitable for a content-based Pub-Sub system. [2]

2) Personalized Interests

Because the subscriber is not coupled with the producer, the users do not have any restriction on making constraints to filter messages so that the user could fully describe his/her interests personally without any concern about the information source.

3) Asynchronous communication between Producers and Subscribers

For the same reason as above, loosed-coupling allows both producers and subscribers to act without concern of each other, both on location and time [2].

Figure 1 illustrates the architecture of how KBN work between Producer and Client.

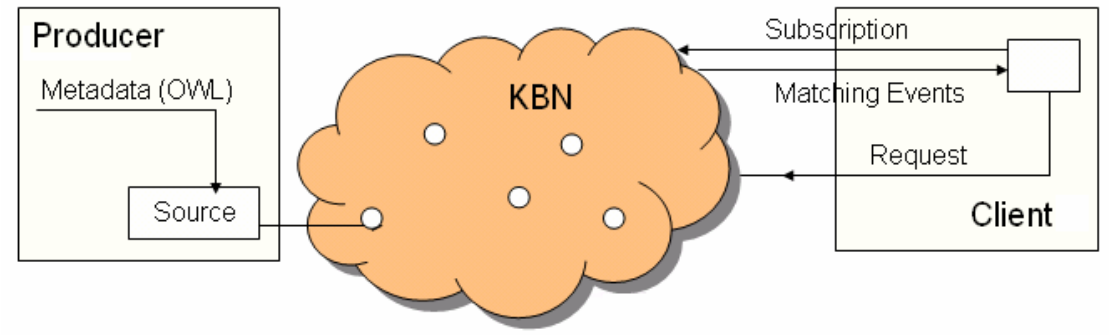

Figure 1 KBN Pub-Sub System Architecture

1.1.3 Instance Selection

There are thousands of ideas to apply the features in section 1.1.1 and 1.1.2 to realistic client applications. Including helping people looking for a property, such as through www.daft.ie, or providing an online market for car businesses, such as www.carsireland.ie and www.carzone.ie. For my project, I decided to develop a Personal IPTV Programme Guide because of following reasons:

1) Many Different Channels with Thousands of Programmes

   To enlarge the audiences figures, lots of channels have published their programmes online, such as RTE and BBC. Also, some client applications [9] are available for user to watch movies, serials, sport games online, such as Tvlinks [10] from UK and PPStream [11] from China. These channels or source providers are located across the internet and the programmes they provide cover thousands of fields. This huge amount of data could make the management difficult for a traditional centralized client-service system. A contented- based routing system would improve the efficiency of message delivery.

2) Different Client Interests

   Each individual viewer has his/her own preference on TV programmes, which are related to his/her age, gender, knowledge background, hobbies, and many other factors. The large amount of TV streams is good, but manually searching a programme of desire could be difficult. The KBN could take the searching work away from user.

3) Asynchronous Behaviors between Channels and Audiences

   The fact is channel make programme time tables without being conscious about audiences' interests, and an audience would not know what time and what kind of programme is on the list. In addition, the publishing action of a channel and the watching action of an audience could happen at any time any place. All these asynchronous behaviors are supported by KBN.

4) Current Problem with Existed Applications.

Currently, there is no such a client application, which integrates all available channels and collects all the programme information. Each channel or application has its own user interface and back-end resources. If a user could not find a specific programme from a current channel or application, he/she has to spend more time to open another one until a valid result is returned. Even worse, the user might lose patience and give up searching. If we make each channel a Notification producer of the KBN, the integration might be achieved.

Another problem is the quality of the results. Searching on single attributes of programme properties could result in the amount of results being too big, in which some content might not updated or not perfectly match personal interests. As discussed above, applying KBN with its filtering ability might improve the result quality for users.

1.2 Research Objectives

In order to make the first steps for the KBN transformation from lab to end client, several objectives were planed to be achieved.

1) To provide a Generic GUI

Currently, the KBN only support command line arguments as input. But the fact is that not every user is a computer expert. A generic GUI should be provided to help users to describe personal interests.

2) To solve current problems with KBN abilities

As mentioned in Instance Selection, section 1.1.3, some current problems of existing IPTV applications might be solved or improved by applying the KBN to a Personal IPTV Programme Guide.

3) To Test Usability of applying KBN into realistic application

The KBN is different from normal content-based networks because of its support for ontological comparison and bag operations. Although the

advantages have been proved [1] [3], the users are not familiar with them. If this new knowledge could not be understood and accepted by users, the advantages might be ignored. The usability should be tested and evaluated with users after implementation.

1.3 Dissertation Roadmap

Seven chapters are organized as below:

Chapter 1 Introduction

Introduce the motivations of this project and a general research object.

Chapter 2 State of the Art

A review on current work of KBN and some background knowledge and technologies related to KBN and IPTV application.

Chapter 3 Requirements and Sources

Specify requirements analysis both on client GUI and application architecture. Also list some available sources, such as technologies and software support,

Chapter 4 Analysis and Design

Analysis and design phase based on requirements, including the design of client GUI and the architecture of entire application.

Chapter 5 Implementation

Presents the process of implementation, including algorithm, problems and solutions. Several screen catches for illustrates the interface of the final production.

Chapter 6 Evaluation

Presents the design and results analysis of a Usability Test Questionnaire. Followed with some reflective evaluation and discussions.

Chapter 7 Conclusion and Further Work

A general conclusion of main contributions in this project. And some discussions on further work.

Chapter 2 State of the Art

This chapter introduces the current state of the art regarding the KBN and some background knowledge and related research and technologies.

2.1 Current work on KBN

As an adapted content-based network, the KBN has all the features of a standard CBN [1]. A content based network keeps the advantages of a Pub Sub system, such as loose coupling, and improves the Pub Sub system by using a message content matching schema rather than using full message type. In turn, the KBN not only has all these advantages, but also supports semantic mark-up which enables the heterogeneity and flexibility needed for autonomic knowledge delivery service.

Currently, the types of the Attribute supported by KBN include Boolean type, Number type, String type and the most challenging ontology type. For all different type of Attribute, there is a set of associated Operators.

A valuable point to note is that, instead of single Value, each Attribute could have its value in a bag type, and each element in this bag could be any one of valid Attribute type.

A primary evaluation in KDEG [1] proves that the performance decrease caused by introducing ontological operators into a CBN is acceptable when the number of hops is limited. The end-to-end time for notification delivery scales linearly with the number of hops increases.

But currently, the ontology comparison in KBNs is only supported for super class and subclass relations.

2.2 OWL and XML Schema

The reason of introducing ontology comparison into a CBN is that a CBN has

a lack of ability to describe the relationship between objects with limited attribute type. Simple reasoning implying “An apple is a kind fruit” or “Dublin is a part of Ireland” is impossible to be represented with integer, string or boolean reasoning [1] [12]. If an ontology is defined as a data model that describes a set of object concepts within a certain domain [12], using OWL, the ontology web language, could enable the representation of an ontology in xml format which has advantages of extensibility, easy to use and easy to query. All these advantages are required in autonomic semantic web services because OWL file is machine readable.

But a problem is that two objects have to be in the same domain, or they are not comparable, such as an apple could not be compared with Dublin city. So some validation should be executed before applying an XML document into real world use. XML schema, which itself is in xml format, is such a high level document used to keep the consistency of content and structure within an XML file [5]. Moreover, the supports on multi namespaces and user-defined complex data type make XML schema more powerful for improving application scalability, flexibility and extensibility [6].

2.3 MPEG and Multimedia Metadata

MPEG, stands for Moving Picture Experts Group, has introduces MPEG 7 as a formal standard to describe the content of multimedia data. MPEG 7 is formally named Multimedia Content Description Interface. Instead of concern about the way the described content is coded or stored, MPEG 7 aims to interpret the content meaning, of multimedia data, into machine readable code. For example, the moment of a football player shooting could be described with MPEG 7 description. Using this description could trigger a record machine to automatically record all moments of exciting shootings during a football match.

Audiovisual data content with MPEG 7 description could be still pictures, graphics, 3D models, audio, speech, video, and composition information

about how these elements are combined in a multimedia presentation [14]. It emphases its aim to benefit as many applications as possible, rather than any individual application.

To further develop the Personal IPTV Programme Guide after this project, MPEG 7 should be attached to support automatic multimedia data processing in different devices and mobile computers.

2.4 Available IPTV Services

Among the available online IPTV applications, most of them provide services in one of three ways: Programme Listing, RSS bookings, and Key Word Searches.

For example, in the up right corner of TV links home page, an online TV website from UK [10], a simple catalogue system classifies the programmes into six groups. Click on one of them, and a programme list is opened with all programmes in alphabetical order. RSS booking is also supported on every page in TV links.

An example for key word searching could be ABC, an online TV station from the United States. The user could search for programmes or shows with some key words in the topic, such as “Prison Break”.

These services are good and very popular with users. The simple catalogue, alphabetically ordered programme list, and key word searching make the searching work easily and efficiently. And RSS booking enables users to keep up with their favourite programmes and interested news without frequently checking manually.

But these services do not fully support user’s personal interests. In addition, RSS feed back returns only when the web site has been updated, but no change in this web site does not mean there is no update in the programme or news. Some updated information might be missed because of the updating between a RSS booking and the corresponding web site.

Chapter 3 Requirements and Sources

In this chapter, the requirements, both on the Client Site and of the entire Application, will be presented. Some primary analysis, based on available resources, will be made with some use cases.

3.1 Requirements

3.1.1 Client GUI

As a client application, a well-designed user interface is very important for the interaction between user and the application. In such personalised guide, the Client GUI is responsible for helping clients to describe their interests as KBN input and clearly present KBN output to the end clients.

But the fact is that the input for the KBN is only supported by command line arguments. All the input has to be in a certain format and follow a set of rules so that the KBN can read and compare it.

However from the view of a user, especially not a computer expert, it is inconvenient to remember all the rules. Also, the number of Constraints increases the risk of typing error using command line input.

Another point should be to point out is that this project aims to apply the KBN features, typically the support for ontological comparison and bag operation. These concepts might be totally new to the end user. The user must have some basic knowledge about these new concepts, or the application could be lost on the user. It is similar to providing a car to a person who does not know how to drive. But how much training is needed at the beginning? It would be reasonable and acceptable for a user to take some time to get familiar with the new application, while it should not be too long and too complex. This is due to the principle that building a client application is to provide convenience and good services.

As a result, the client GUI needs to achieve the following general requirements:

1) Reduce the skill and knowledge required from a client
2) Maximum the ease of use of the client
3) Automatically translate user input into the format that accepted by KBN
4) Clearly present the KBN matched resulted to client

3.1.2 Application

Although the background work of the Client GUI is hidden from users, it is another important part that influences the service quality, including efficiency, flexibility and extensibility. The general requirements should include:

1) The Personal IPTV Programme Guide itself should be easy and cheap to maintain
2) As a first step for the KBN transformation from lab to users, the application should maximise support for the KBN features and present the benefits of the KBN, including ontological comparison and the bag concept [2].
3) As one of the client applications for the KBN, it should enable the TV programme guide to switch to another client application with minimum costs and time. In order to achieve this, the GUI should be separated from the application beneath it.

3.2 Available and Required Resources

As an extended application of the KBN, this project will use a Java version of the KBN which is provided by Dr. John Keeney. In order to finally integrate with the KBN, all the functions required would be developed and implemented in Java within the eclipse IDE. The following technologies and knowledge might be useful during the implementation:

1) Java Swing - Client GUI

Java Swing is a graphical user interface toolkit for Java, which contains combo boxes, buttons, and lots of other widgets for building java based application GUIs[17] [18].

2) Jena - Ontology Handler

Jena is a Java framework for building Semantic Web applications. In this project, Jena will be used as the ontology handler to parse an OWL file so that the tree structure of the concepts defined in this OWL file can be discovered and documented.

3) Java DOM - XML File Handler

Java DOM is a part of the Java API for XML processing. In this project, DOM is used rather than SAX, because Document Object Module views an XML file as a group of objects with their internal relations. It supports free navigation and enables accessing and manipulating on each object. [20][21][22]

4) Exsiena Packages - Integrate with the KBN

Exsiena Packages, provided be John Keeney, include all KBN interfaces in Java format. All the communication between the client GUI and the KBN are supported by these interfaces.

Chapter 4 Analysis and Design

This chapter specifies the details of the analysis and design phase, including the design of the client GUI and the architecture of the entire application.

4.1 Client GUI Layout Design

As mentioned in requirements, section 3.1.1, the Client GUI needs to handle the client input and clearly present the result. So that GUI would be split into two parts: Subscription Generator and Result Area. Because the main goal is to allow the client to use the KBN features, more emphases would be put on the Subscription Generator section in this project.

Another point I need to always remind myself is that this Programme Guide is just one instance of a client applications for the KBN. Considering extensibility and flexibility, the client GUI design should be independent and not too much rely on the Metadata of a specific TV stream.

4.1.1 Subscription Generator

The Subscription Generator is the most difficult part of this project, because it includes lots of new concepts which are not familiar with the client. On one hand, I need to fully use the KBN; on the other hand, I need to consider the user receptivity and the application practicability. I analyse and design this Subscription Generator in the order that firstly design the layout of the general view, then details each individual component, and finally place all components as designed layout.

4.1.1.1 Subscription and constraints

In a KBN Subscription, there is a filter which contains the client interests. This filter has a certain format that is as follows:

filter { Attribute1*Operator*Value Attribute2*Operator*Value …}

Each combination of Attribute*Operator*Value is called a Constraint. Between adjoining Constraints, there is one and only one space.

Based on this structure, the general view of the Subscription could be designed as in Figure 2:

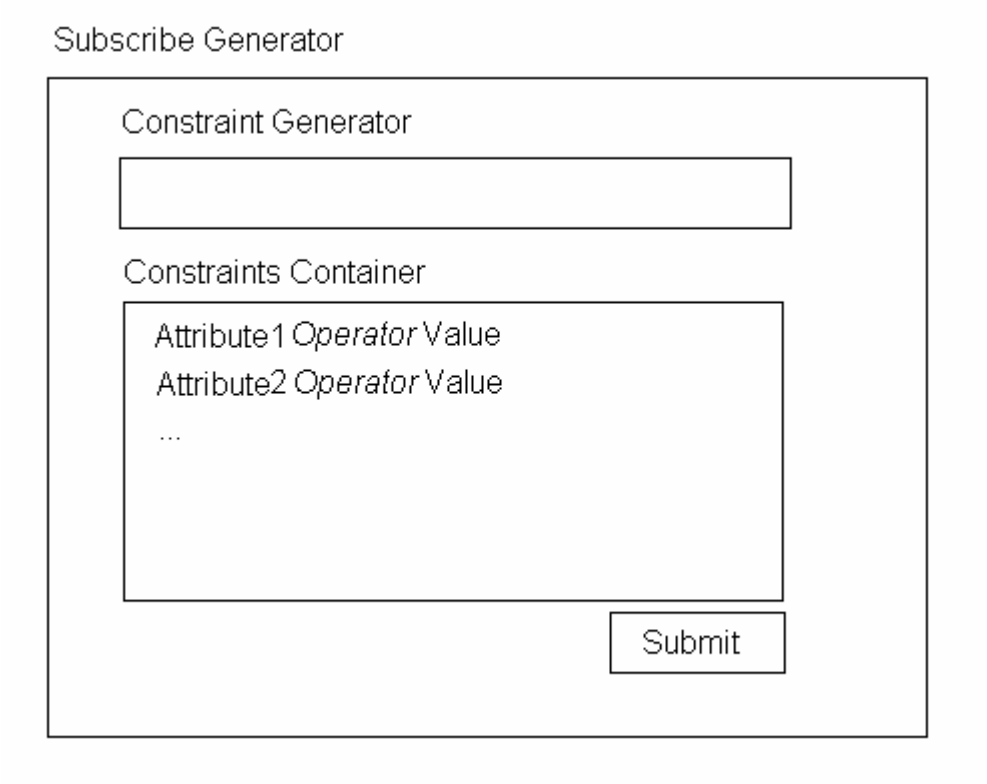

Figure 2 Subscribe Generator Layout

Obviously, the missing area, the Constraint Generator, is the most complicated part of the Subscription Generator. The analysis and design of this area will be detailed in next section.

4.1.1.2 Constraints Generator

The Constraints Generator is at the core of the client GUI. A general workflow is shown in Figure 3.

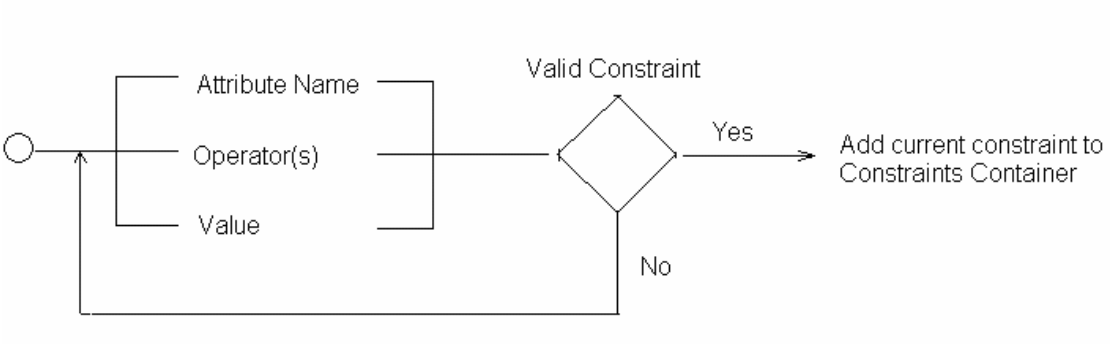

Figure 3 Workflow of Constraint Generate

1) The rules of a valid constraint

   a) One rule is a valid constraint must consist of three parts: AttributeName, Operator type, and Value. As can be seen in the filter format that the order has to be Attribute, Operator, then Value.

filter { Attribute1*Operator*Value Attribute2*Operator*Value …}

To follow this structure, the Constrain Generator would used to be separated into three main areas and an “Add” button placed beside these input areas for adding the new constraint into Constraint Container (Figure 4).

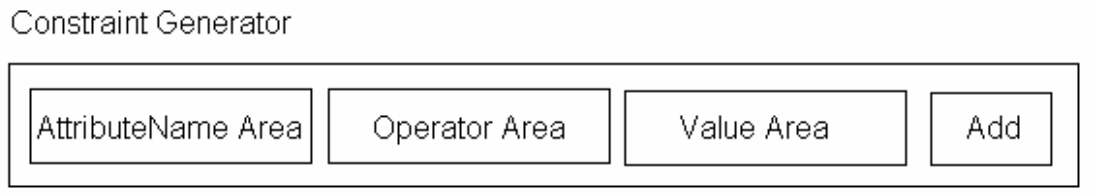

Figure 4 Constraint Generator Layout

There are two advantages to this design.

First, the users is more likely to fill in blanks. If there are three empty areas, the users would quickly understand that there are three

variables need to be specified.

Second, the users do not need to care the order of three components. This is because the Client GUI will take the user input, translate it and order it until it tallies with the structure of a valid constraint. The translation of user input will be discussed in next section, 4.2 Terminology Present.

b) The second rule is that the Operator and Value have to follow the type of the Attribute. This means a Number type attribute has to be specified with an operator, which is one of the [”>”,”<”,”=”,”>=”,”<=”], and the value has to be a valid number. Also a string type attribute has to be followed with an operator which is one of ["=",">*",*<","any",!=","*"]. For example, if StarRating is an int type, then

StarRating > 1	Valid,
StarRating > 1.0	Invalid, 1.0 is not an int type
StarRating < “1”	Invalid, “1” is a string type
StarRating >* 1	Invalid,>* is a string operator

Likewise, if Title is a String type attribute,

Title>* “MR”	Valid
Title *< MR	Invalid, MR has to be round with quotation marks.

A more complicated case is the ontology type attribute. If the attribute “Country” is an ontology type and defined in Country.owl, the value has to include the name space as a prefix, such as

Country #=”http://www.owl-ontologyies.com/unnamed.owl#Ireland”

And importantly quotation marks are required.

This rule requires a user to remember:

i) The type of a certain Attribute

ii) The suitable operators for use with a certain type

iii) The Number type has to be a number value in the valid range and without quotation marks, while quotation marks are required by String or Ontology type Attributes

iv) The namespace of an ontology type attribute, which could be changed for each individual attribute.

The problem is the first three items require too much knowledge and skill of the users, and for the last namespace prefix, it is near impossible for a user to always keep updated with which OWL file is being used and which namespace is applied, as the application could be updated and changed in real time.

In order to provide service and convenience to the users, the content of operator area and value area would be dynamically load for the user.

First, the operator area would use a combo box, in which the valid operators would be dynamic loaded for a user to select.

Second, user input validation would be executed before the constraint is added into constraint container. An alert message would be returned to help the user to understand the reason for an invalid constraint.

c) Alert Message

As an error prevention measure, an Alert Message would be designed into the Subscription Generator to alert the user about what makes the constraint invalid. The Client GUI is responsible for sending an Alert Message in the following two cases:

Firstly, any missing value in the input box of Attribute, Operator or Value.

Second, the value of a Number Type Attribute is out of range, such as the input is not a number and could not be cast into a Number Type.

The Alert Message would be placed under the Constraint Container.

d) Undo and Redo

Although all the new constraint would be checked, it is possible for a user to add mistakenly an unwanted but valid constraint. Based on Nielsen's Heuristics [3], an "emergency exit" should always be provided to support user control and freedom. A "Remove" button was added into the Subscription Generator to support undo and redo.

4.1.2 Result Area.

Because the center of gravity is on the Subscription Generator, the design of the Result Area would follow the principle of making it clear and simple. But before designing the layout of the Result Area, one issue which should be addressed is the management of the subscription.

4.1.2.1 Subscription Management Concern

If there are only one or two subscriptions, the user could recognise the subscription from the result quickly. But as the number of subscription increases, this may become a problem.

A Subscription Serial Number could be a simple solution. This serial number is automatically generated by the Client GUI when the user submits a new subscription. If it is the first subscription the client made, the serial number would be S0, and the second subscription would be named S1.

This serial number schema is a temporary schema. As a client application, It should allow the user to name the subscription with realistic meaning instead of a simple number. The improvement of subscription management would be included in the future work section.

4.1.2.2 Result Area Components

There are only three components included in Result Area. There include:

a) Result List

Each matched result would be added at the bottom line in the Result List area as one record. And each record would contain two parts: subscription serial number and the metadata of the matched result. Each matched result would be displayed as

S0: {Attribute1 = value Attribute2 = value….}

b) Clear Button

When there are too many old results, a “Clear” Button is placed below the Result List to clear the list.

c) Corresponding User Subscription

With the simple subscription serial number, the user might not remember the original subscription. And based on Nielsen’s Heuristics[3], a well designed client application should not ask the user to remember information from one part of the session to another. In order to make the contents of original subscription retrievable and visible, the corresponding subscription will be displayed under the Result List when the user clicks on a certain result with a prefix of the serial number.

Based on all the design above, the final Client GUI layout would be as Figure 5 on next page. And the detailed design of an individual area, such as Operator Area and Value Area will be analysed in next section.

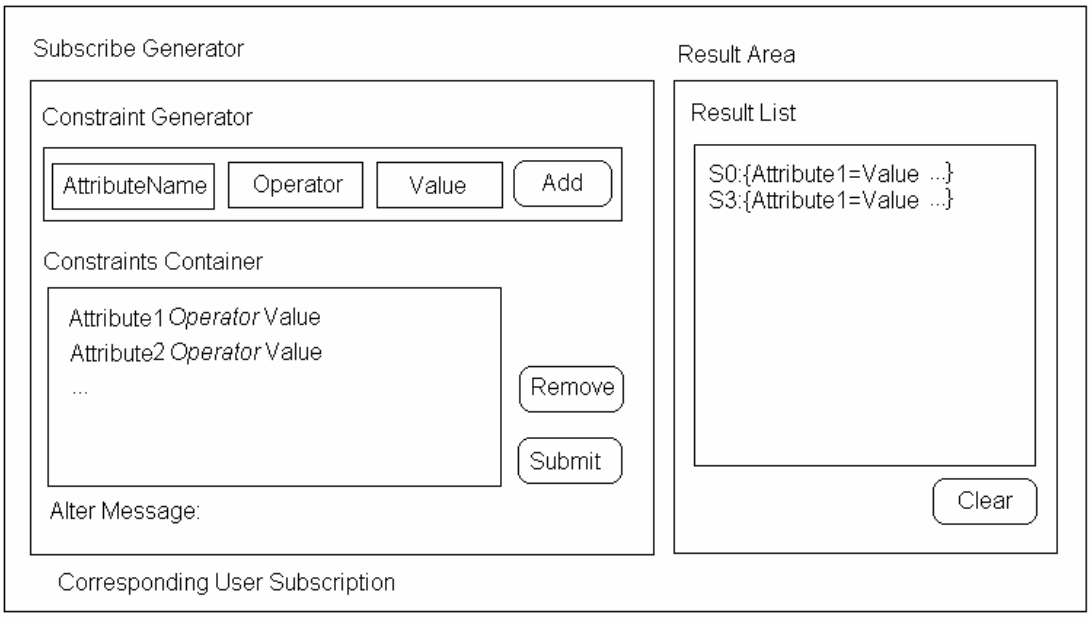

Figure 5 Client GUI Layout

4.2 Terminology Presentation

Currently, all the users of the KBN are Computer Science experts who are familiar with the terminology used in KBN. But the Client Application is developed for the general public, this terminology becomes an obstacle. Moreover, the ability of the supported KBN does not promise it will have a realistic meaning, such as “any”, a number or string operator.

Attribute1 *any* “some string value”

The above expression will cause the KBN to return all Notifications with a field “Attribute1”, because “any” means Attribute1 has any string value. This disables the filtering ability, which is one of the main features this application wants to utilise.

This section discusses the terminology translation. On one hand, the operator symbols should be translated for the users to use the KBN easily, and on the other hand, the user input operators need to be translated into formal symbols

for KBN to reason. The design phase of terminology translation will analyse using examples. Because the Bag and its Composite Relations [2] is a more complicated case, the analysis and design would be separated into Section 4.2.3.

4.2.1 Basic Operator Selection and Translation

Operators are one of the three indispensable fields in a valid constrain. Here Basic Operators includes all the valid operators for Number type, String type and Ontology type. As well as Bag Operator which will be discussed later because of their its unique features.

As introduced in layout design, using a specific operator is based on the type of the attribute, therefore the basic operators are organized into three sets, as follows:

Number Type Operators

"=" EQUAL

"<" LESS THAN

">" GREATER THAN

">=" GREATER OR EQUAL

"<=" LESS OR EQUAL

"any" ANY NUMBER

"!=" NOT EQUAL

String Type Operators

"=" EQUAL

">*" HAS PREFIX e.g., "software" >* "soft"

"*<" HAS SUFFIX e.g., "software" *< "ware"

"any" ANY STRING

"!=" NOT EQUAL

"*" SUBSTRING e.g., "software" * "war"

Ontology Operator

"@=" EQUIVAL e.g. Apple @= Apple

"@>" MORESPEC or EQUIVAL e.g. Apple is more specific than Fruit

"@<" LESSSPEC or EQUIVAL e.g. Fruit is less specific than Apple

When the attribute type has been reasoned, the items of the operator combo box are dynamically loaded from the corresponding operator set. As explained, the “any” relation does not have any practical meaning, it is taken out from both the Number and String Operator Set.

But in fact, the final version of the implementation is different from the initial design. In the initial design, the items of the operator combo box would be loaded as shown in Figure 6 Initial Design of Operators.

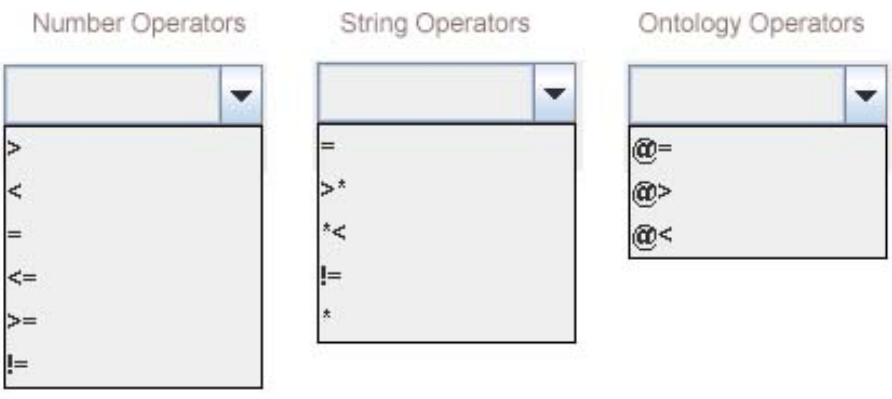

Figure 6 Initial Design of Operators

These symbols are used for two reasons. First, they are simple and terse. Second, they are already supported by Extsiena, which is the java version of the KBN interface and provided by John Keeney. Extsiena would accept the above basic operators as a string input and then translate them into a short type value which is supported by the KBN.

This presentation was has been changed in the final version. The reason for this is, as reminded several times above, the application is developed for users. An easy abbreviation for KBN experts could cause troubles for public users. Therefore, shown in Figure 7, the operator symbols were replaced by natural language, which is familiar to the users. Because the number type operator has already been wide accepted in daily life, it was not changed.

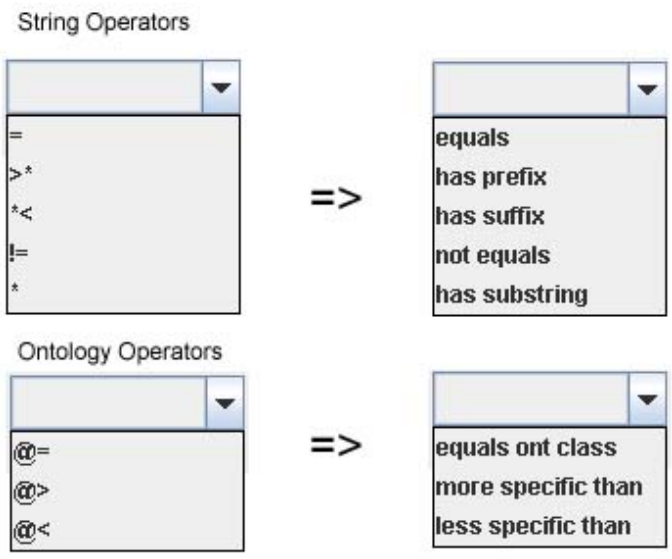

Figure 7 Changed Operator Symbols

As a result, the Client GUI needed to be able to translate user input the specific back into operator symbol and the symbol will be translated into a short value by extsiena in turn so that KBN could accept and read it.

4.2.2 Value Type Presentation and Translation

The easiest way to ask for user input is a text area in which the user can type in whatever they want. But if there are some constraints on user input, a text

area could cause lots of problems. A good example of value area design is the date input widely used in various thousands of client applications. The month value would always be a number between 1 to 12, which enables easy use and reduces the risk of a typing error..

4.2.2.1 Number Type and String Type

Although the disadvantage of using a text area has been discussed, it has to be used for the Number type and the String type Attribute in this application. One reason is that it gives freedom to express, which is a main aim when users describe personalized interests. Another reason is, unlike the date, there is no fixed range of values which could be used for checking.

Fortunately, the use of a text area here would not cause lots of problems. For the number type, the validation check would be executed before adding a new constraint. And for the String type, the users do not need to worry about the quotation marks because it is already a string type value by default.

4.2.2.2 Ontology Tree

The presentation of the ontology type value is one of the most import parts in the design phase. This is because the ontology is a totally a new concept to most public users, how to make it easy for the user to understand is the key task. What follows are some factors which need to be addressed:

1) In Section 2.1, it has been discussed that, currently, the KBN can support ontology reasoning only with super class and subclass relations. It is suitable for discover “is part of” relationships, such as “Dublin is part of Ireland”, or “is a kind of” relationships, such as “Apple is a kind of Fruit”. Indent format is a good choice for present these kinds of super class and subclass relations. Such as:

Food

->Fruit

-> Apple

->Orange

->Meat

Clearly, “Fruit” and “Meat” is the subclasses of Food, and “Fruit” is the super class of “Apple” and “Orange”.

2) A selection option is better than free text. This is because the class specified from the input has to be exactly the same as defined in a certain OWL file, and then the ontological comparison can be executed. It is impossible for users to remember all the classes and there is no guarantee of the spellings correctness.

3) Moreover, the number of classes defined in one OWL file could be unlimited or limited. It is better to allow user to set the visibility of the classes.

4) Lastly, some information necessary in OWL files, such as namespace, might be meaningless for a user in describe his/her interests. Following Nielsen’s Heuristic [3], the irrelevant information should be maximum diminishes from users visibility. Therefore, unrelated information, like namespace, should be hidden from users.

Considering the above factors, a tree model is the best used to present the structure of super class and subclass relationship defined in an OWL file. Supported by Java Swing, a tree has indent modality and each node which is not a leaf could be collapsed or expanded. And at each node, information could be stored for KBN query but only relevant class names displayed for the user to select.

4.2.3 Bag and Composite Relations

The Bag and its composite relations is separated because its particularity and complexity. As implemented by Dominic Roblek in his NDS project last year [2006], the KBN can support a bag using a comparator matching algorithm [2]. According to Dominic Roblek, there are three binary bag relations:

Bag Operator

"#=" EQUAL BAG e.g. [1,2,3] equals to [1,2,3]

"#>" SUPER BAG e.g. [1,2,3] is a super bag of [1,2]

"#<" SUB BAG e.g. [1,2] is a sub bag of [1,2,3]

But what would be applied to this project is the composite relationships of bags which greatly extend the expressiveness of the KBN subscription matching mechanism. The composite relation can be simply described as a combined use of the bag operator and the subbag operator, which could be one of the basic operators. Bag operators describe the relationship between bags, and the subbag Operator, which depends on the type of the bag, describes the relationship between the elements in the bags. To assume users understanding, some example could be:

Three basic types of bag:

Int bag: [1,2,3]

String bag: [“abc”,”defg”,”hijk”]

Ontology bay: [“http://www.owl-ontologyies.com/unnamed.owl#Apple”,

”http://www.owl-ontologyies.com/unnamed.owl#Orange”,

”http://www.owl-ontologyies.com/unnamed.owl#Banana”]

if all above classes are defined in an OWL file, such as Fruit.owl

Take int bag as an example:

[1, 2] #< < [1, 2, 3] because 1<2, and 2<3

[1, 2, 3] #> > [0, 1,1] because 1>0, 2>1, and 3>1

The following expressions are wrong:

[1,1,1] #< < [1,2,3] because 1<2,1<3, but the last 1 not less than 1

[1,2,3] #> < [2,3,4,5] because [1,2,3] is not a super bag of [2,3,4,5]

[[1],[2],[3]] #= = [1,2,3] because the type of bag is different

4.2.3.1 Bag and Composite Relations Selection

Theoretically, the bag composite relations make the KBN so powerful that the subbag of a composite relation could be a composite relation itself. Such as the last expression above, [[1],[2],[3]] is supported in the KBN as a bag of integer bags.

But for this Client TV Program Guide, implementing all these relationships are too complicated for the users and might lose the application realistic meaning. In order to keep it simple as well as powerful, the following concept would be implemented in this project for some practical reasons as the metadata for a TV stream:

1) Three types of bag

   Int bag could be used as date or time value: [1984, 5, 6], [17:54]

   String bad could be reasonable for actors list: [“Brad Pitt”, ”Angelina Julie”]

   Ontology bag would be used to describe some logical knowledge in a range: geographical knowledge [“Ireland”, “England”,”South America”]

2) The element in one bag could only be a single value of int, String, or ontology type.

   If “Travel” and ” Sports” are the class defined in Catalogue.owl, and “Ireland”, “England” are defined in Country.owl, the following expression:

myInterstes #</(#</@>) [[“Travel”,” Sports”],[“Ireland”, “England”]]

could be used to describe the user want to know all the Travel or Sports information from Ireland or England, Although this is feasible, it could be achieved with two constraints:

InterestedCatalogue #</@> [“Travel”,” Sports”]

InterestedCountry #</@> [“Ireland”, “England”]

In this approach, the expression is simpler and easier for user to handle with the same effect.

4.2.3.2 Bag and Composite Relations Presentation Design

In order to keep the application consistent, the input mode would be kept measure for each single element in a bag, using text area for number and String type and using tree selection for ontology type.

The next concern was the presentation of a bag.

For number or String type, the most common way to describe several values is using a comma to split individual elements.

For ontology type, because each single element is selected by user input, splitting them with a comma is not suitable. One approach is “multi selection” supported by the Java Swing Tree Model. The problem is that the selected item could be lost unconsciously by a user’s mistake, such as clicking on some empty irrelevant area. Another approach is to ask the user to select one ontology class each time and add the ontology class to a list. This approach is better for three reasons.

Firstly, the lost problem of multi selections could be avoided.

Secondly, the list presentation suits the concept of a bag. When a user triggers an add action for an ontology class, it simulates the action of put one thing into a bag.

Thirdly, the add action may be familiar to most users because some email

systems use similar ways to support sending a mail to a group of users by inserting the address from a saved contact book, such as Hotmail or Yahoo.

Based on the above analysis and two operators would be involved in a bag composite relation. The final Constraint Generator might be changed for a bag into a view like Figure 8:

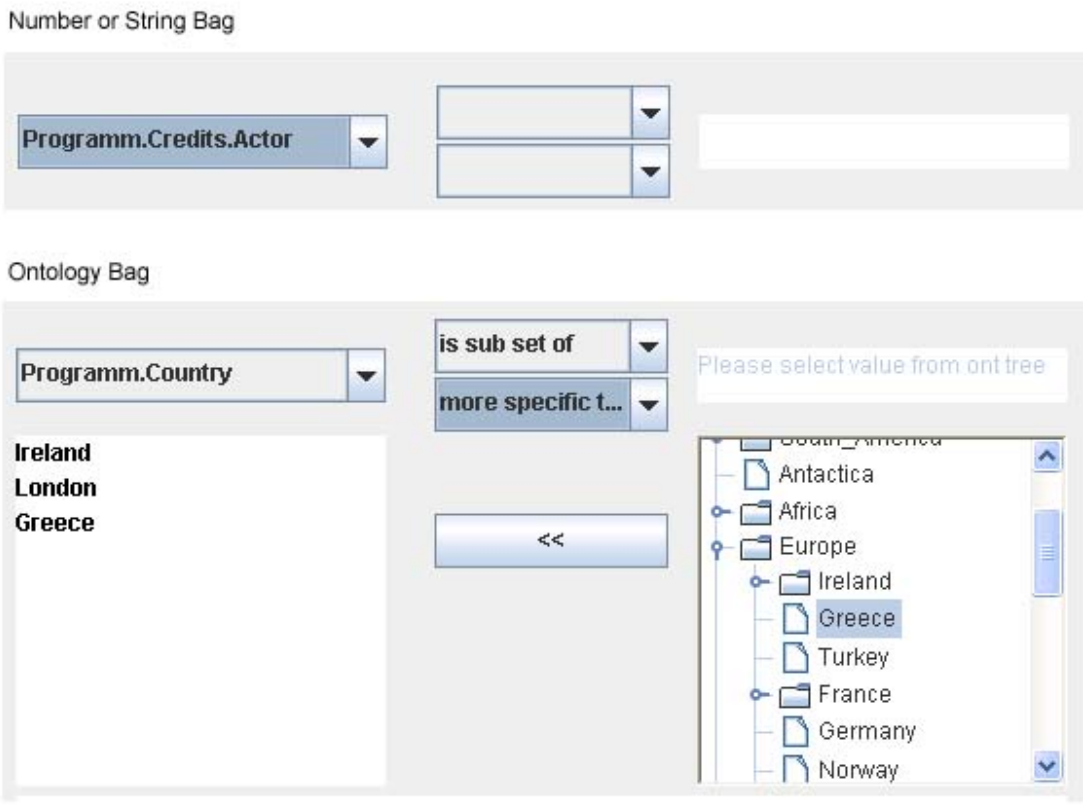

Figure 8 Constrain Generator for Bag Type Attribute

4.3 Application Architecture

This section analyses the architecture of the client application for the KBN. The ingenious use of XML schema, an xsd file, will be discussed followed by the class diagram design.

4.3.1 Multi Benefits of Using XML Schema

In this client application for the KBN, the only connection between the client

GUI and the background application is an XML schema file. This xsd file is used in a smart way to support the application's consistency, flexibility, scalability, extensibility, and independency.

4.3.1.1 Effect on individual client application for KBN

For an individual client application for the KBN, such as this project, Personal IPTV Programme Guide, the XML Schema Definition could be used to:

1) Keep Consistency between Producer side and Client Side

   For a distributed service built on top of the KBN, there is a precondition that the metadata attached to a Notification and the filter contend in Subscription has to be comparable. The comparability means:

   a) The attribute name has to be exactly same, including order and capital case.

      "Title" is not equal to "title", because KBN is case sensitive.

      "Catalog" is not equal to "Catalogue", although it is the same meaning in human interpretation.

   b) The type of the attribute has to be same.

      Notification: x = 3 vs. Subscription x>"3" is not comparable

      because 3 is an int type, while "3" is a string type.

      Notification: x=5 vs. Subscription: x=[5] is not comparable,

      because a single value could not be compared with a bag type.

   c) For an ontology type, the value must be defined in the same OWL file

   Notification: Catalogue = "Apple" vs. Subscription: Catalogue@="Fruit"

   It might not be comparable when both "Apple" is defined in RoundShapeObject.owl, while "Fruit" is defined in Food.owl

   To keep the consistency to enable comparability, the Producer and the

Client should share the metadata naming schema and the relevant data type. An easy way to achieve this is to defined a name and the data type for each leaf element in the XSD file. And also specify the URI of the OWL file for the ontology class.

Int type: <element name=”VideoDuration” type=”int”/>

String type: <element name=”Title” type=”string”/>

Ontology type: <element name=”Catalogue” type=”string”

source=”file:..Catalogue.owl”/>

Bag type(an ontology bag): <element name=”Actor” type=”bag”

bagtype=”ontology” source=”file:..Country.owl”/>

With this XML Schema, the consistency could be guaranteed, because both Notification from producers and Subscription from clients would be check against this XLM Schema.

On one hand, metadata from Producer could be validated before publishing. On the other hand, at the client side, GUI would do the following steps to match the attribute properties with Producer:

Step 1: Dynamically load these elements into an Attribute Name List for the user to select.

Step 2: The selection of one attribute would trigger the discover action of the attribute’s properties, including type, source and bagtype if necessary.

Step 3: Based on these properties, the operator combo box would be loaded for the right type.

Step 4: The value area would change for the type of attribute, and the ontology tree would be generated as required.

2) Support Flexibility of Service Management

Because of the support for complex data types by XML schema, such as user defined types, the service provider could change this XSD file as. Take this Personal IPTV Programme Guide as an example, the data types

of Programme and Movie could be defined as following to support normal daily service.

```
<complexType name=”Programme” />
    <element name=”Topic” type=”String”/>
    <element name=” Emcee” type=”String”/>
    <element name=” Catalogue” type=”ontology”
                                    source=”file:./Catalogue.owl”/>
</complexType>
<complexType name=”Movie” />
    <element name=”Title” type=”String”/>
    <element name=” Participator” >
            <element name=” Director” type=”String”/>
            <element name=” Cast” type=”bag” bagtype=”string”/>
    </element>
    <element name=” Catalogue” type=”ontology”
                                    source=”file:./Catalogue.owl”/>
</complexType>
```

The convenience of service management could be presented as a case of Olympic Games. When the Olympic begins, the service provider could add an OlympicGame data type into this XSD file, because the query of the games would increase during the Olympics.

```
<complexType name=”OlympicGame” />
    <element name=” Catalogue” type=”ontology”
                                    source=”file:./SportCatalogue.owl”/>
    <element name=” Country” type=”ontology”
                                    source=”file:./Country.owl”//>
```

<element name=” Participator” type=”bag” bagtype=”string”>

</complexType>

After the Olympics end for a period of time, this data type could be deleted because of the transient factor of Olympic Games.

In addition, if the survey shows that the “Director” in “Programme” should be a string bag like “Cast”, or the OWL file has been updated, the service providers do not need to ask the end client to update their application, instead they just need to change the XSD definition. For the ontology updated, still no changes need to be made, because it is still referenced by the URI, from which the classes would be loaded dynamically.

The independency between the client application and service context ease the performance of the service on management, updating and maintenance.

3) Enhance the Scalability of Application

In the above examples, the “Catalogue” in “Programme” and in “Movie” shares the same name and the same OWL source. This might cause problems for the KBN during comparison.

Fortunately, XML schema supports namespaces to solve the problem of name confliction, this has already been discussed in the KBN Naming Convention[8]. If developed across a large scale area, such as the internet, namespace support will be a good addition to improve the application scalability.

Due to time restraints, namespaces have not been applied in this project.

In order to solve this name confliction problem, another approach is that the GUI would attach the parent elements name as prefixes to distinguish the attribute name. This will be detailed in the Implementation section.

4.3.1.2 Effects over different client applications for KBN

As mentioned in the Requirements, chapter 3, this project is one instance of the client applications for KBN, IPTV is just concerned with the metadata design.

The independency discussed above not only improves the service management for an individual application, but also enable the extensibility and flexibility when switching between different client applications.

The independency overview of client application architecture could be drawn together as shown in Figure 9.

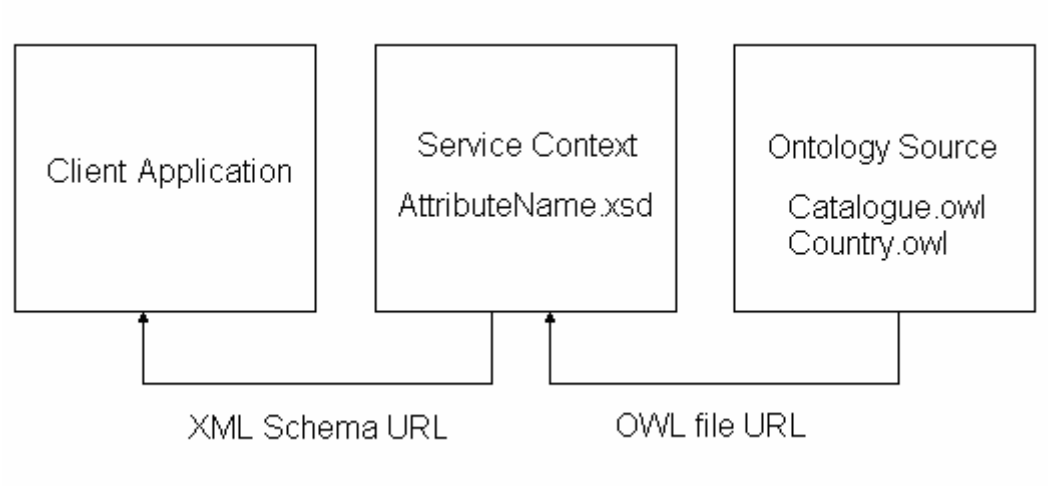

Figure 9 Independency between Application, Context and Source

4.3.2 Class Diagram

To efficiently implement this Programme Guide, the classes would be designed with Object Orient Design principles, as documented below.

There are four main classes that would be needed:

1) Graphical User Interface
    a) Responsibility for building the application interface for users
    b) Guide users in describing their interests in terms of constraints and subscription
    c) Send the subscription to KBN
    d) Present result to users
2) Attribute Name List Loader

    Once the application starts, load the service context defined in AttributeName.xsd into Attribute Name List for the user to select.
3) Ontology Tree Drawer

    Draw the ontology tree for users to browse and select.
4) Ontology Type Bag Drawer

    Provide an empty bag and load the ontology tree for users to specify the ontology type bag value.

The Java Bean model would be applied for data passing and sharing within the application. More details would be included in the Implementation section.

The Class Diagram for main functional classed is shown in Figure 10 in next page.

Figure 10 Class Diagram

Chapter 5 Implementation

This section presents the process of implementation, including algorithmic decisions, problems and solutions. Final production will be illustrated with several screen catches of the application in operational mode.

5.1 Client GUI

Java Swing is used to build the client GUI. Both visible and invisible components are used. Visible components include combo boxes, text areas, buttons, trees, lists and labels. Invisible components include Frames, JPanels, action listener and list cell renderers. All these components are used for certain reasons.

5.1.1 General Interface

As shown in Figure 2, section 4.1.1.1, page 15, the overview layout of the GUI is very simple so that all the position and size of component are specified within the .setBounds(x ,y, width, height) function. The general interface at the start is shown as Figure 11 on next page.

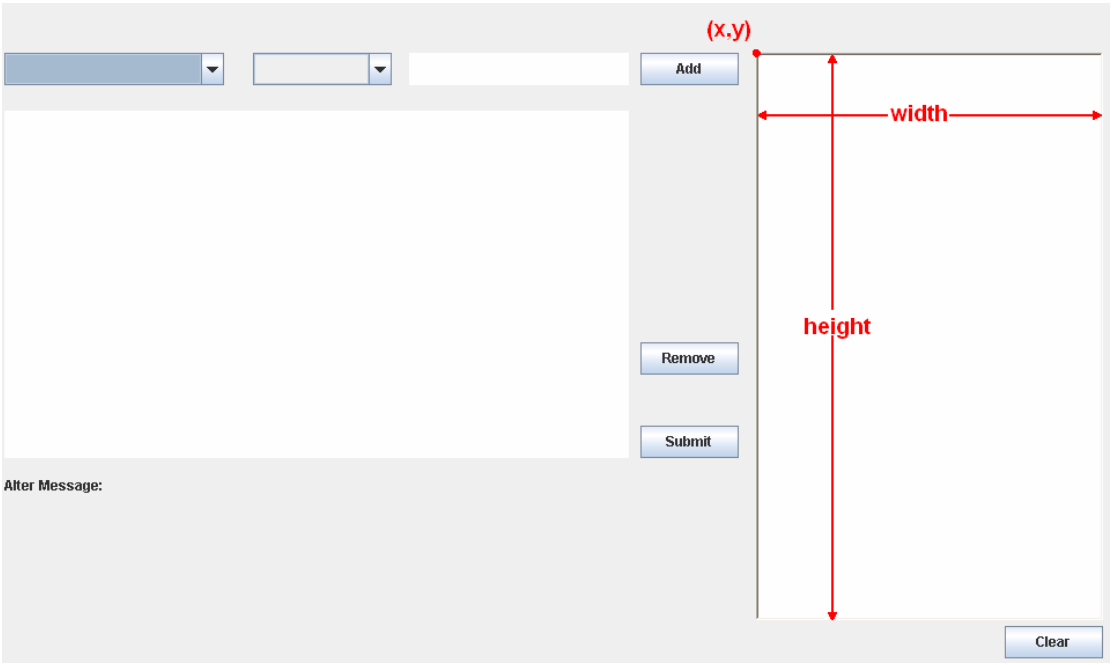

Figure 11 General Client GUI at Start Time

5.1.2 General Constraint Generator

From a general view, the development Constraint Generator is needed to implement user guide, constraint validation, alert message reminders and the default reset.

1) User Guide

This is implemented by adding an item listener into the Attribute Name Combo Box. Once the user selected an attribute, the listener would trigger the loading of the operator combo box and the change near value area.

2) Validation, Alter Message, and Reset

These are all achieved by adding an action listener to the “Add” button. The following steps would be executed when a user clicks the “Add” button:

Step1: Check missing filed, including three fields for a single value attribute and four fields for a bag attribute. Alert message would be returned and the execution breaks. Some examples are illustrated in Figure 12 &13.

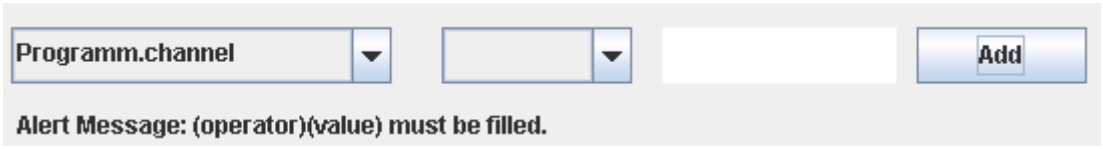

Figure 12 Invalid Constraint Case 1: String Type Constrain

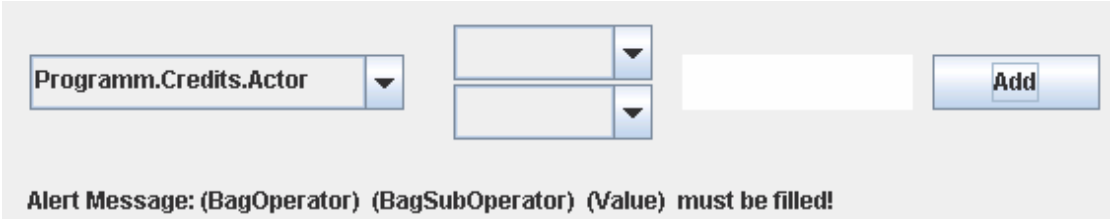

Figure 13 Invalid Constraint Case 2: String Bag Type Constrain

Step2: Check the number data type for validity. If a Number Format Exception is caught, alert message would be returned and the execution breaks. Some examples are illustrated by Figure 14.

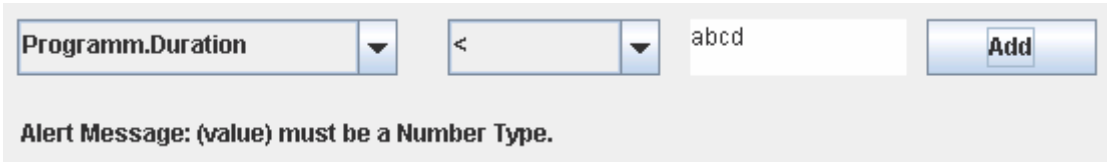

Figure 14 Invalid Constraint Case 3: Invalid Number Data Input

Step 3: If the constraint passes the validation, it would be added into the constraint container, and all settings would be restored to their default values, as shown in Figure 15.

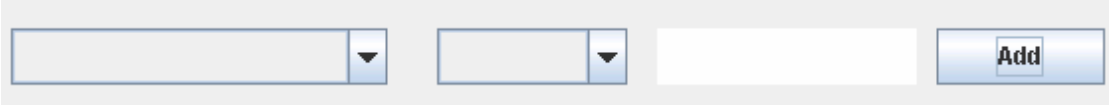

Figure 15 Default Setting of Constraint Generator

5.1.3 Operator(s)

As explained above, a valid constraint add action would trigger the automatic loading of operator(s) combo box.

Some pre-work is necessary to define four sets of operators as String arrays at the application start-up.

String[] numberOperator = { ">", "<", "=", "<=", ">=","!=" };

String[] stringOperator = {"equals", "has prefix", "has suffix","not equals", "has substring" };

String[] ontologyOperator = { "equals ont class", "more specific than", "less specific than" };

String[] bagOperator = { "equals set", "is super set of", "is sub set of" };

When loading is triggered, the following workflow is execuled:

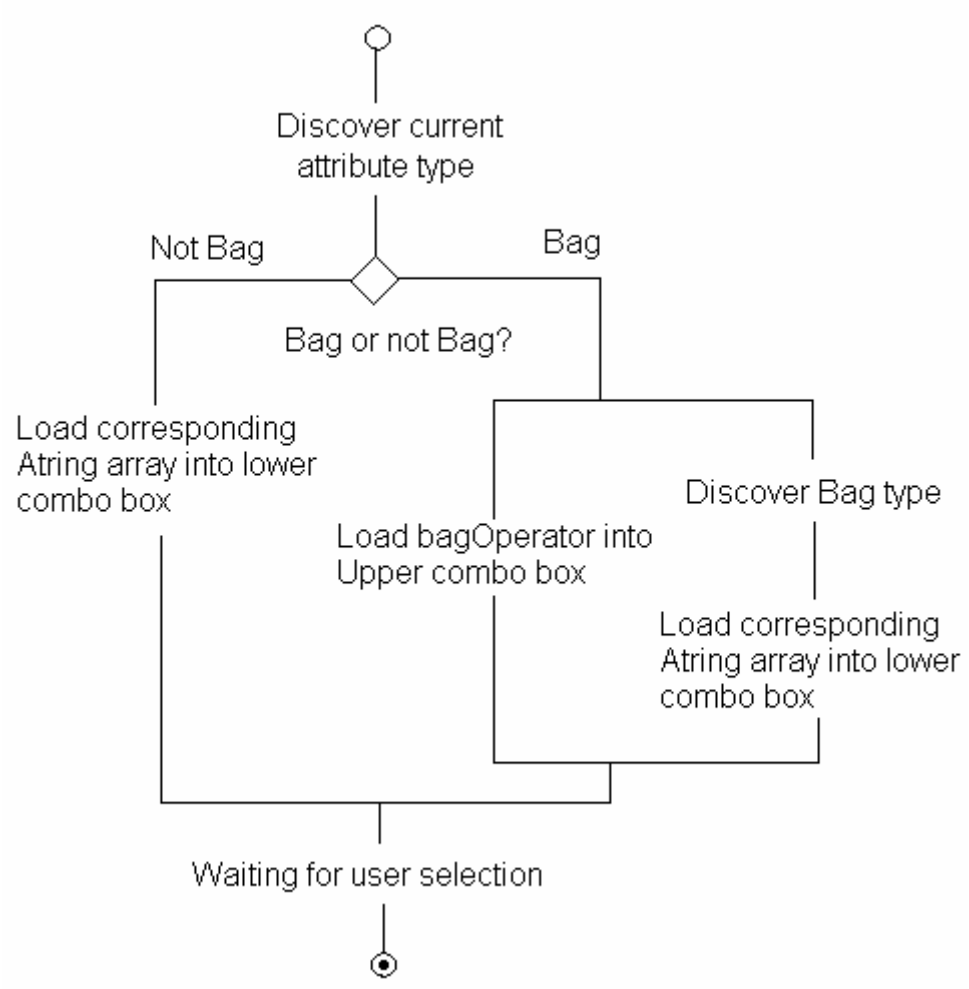

Figure 16 Workflow of Loading Operator(s)

Only one loop is needed to load the items from the string arrays into

combo boxs.

5.1.4 Ontology Present

To build the tree from an OWL ontology needs the cooperation of Jena and Java Swing. When Jena find a class, the tree model adds it as a tree node.

The most difficulty aspect is the discovery of the tree structure of the OWL file and creating a corresponding class as a tree node.

The ontology tree is built from an OWL file, therefore the first step is to use Jena to read the OWL file. However Jena is not powerful enough to understand the entire tree structure at first glance, it is only capable to find out the root and find out the subclasses of the current class. As a result an algorithm is required to help discover the tree structure.

The first plan was a for() loop. Unfortunately, this approach fails very soon. The loop times could not be determined because the tree depth is unknown.

The second attempt was a while() loop. But this fails too. After it successfully finds the first leaf node, the loop stops so that the entire tree can not be discovered.

The final solution is a depth first algorithm using a recursive method. A large while() loop is used to discovery the first level children of the root, then the recursive method is called on each node until the leaf node is detected.

```
public void showHierarchy(PrintStream out, OntModel m) {
        Iterator i = m.listHierarchyRootClasses().filterDrop(new Filter() {
        public boolean accept(Object o) {
                return ((Resource) o).isAnon();
                }
        });
// the while() loop under tree root.
        while (i.hasNext()) {
```

```
                OntClass oc = (OntClass) i.next();
                DefaultMutableTreeNode subroot = new
                                DefaultMutableTreeNode(oc.getLocalName());
                root.add(subroot);
// the recursive method
                showClass(subroot, out, oc, new ArrayList(), 0);
                }
        }
```

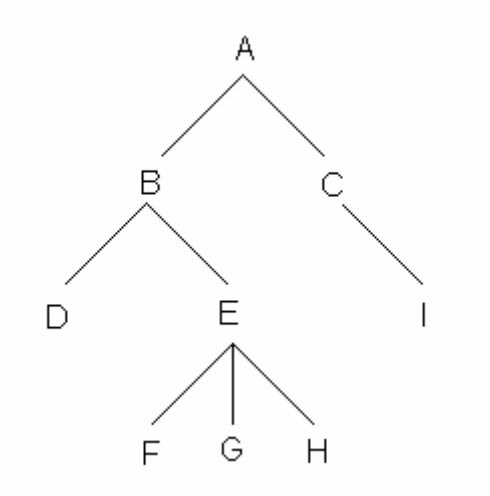

Figure 17 Tree Structure Sample

Based on this algorithm, the discover order of the tree structure in Figure 17 would be A,B,D,E,G,G,H,C,I.

In this approach, Jena is used to discover the sub class of the current ontology class, and a tree model is built from the ontology classes.

5.2 Application Architecture

5.2.1 Loading Service Context to Client GUI

In the client GUI, the Attribute Name List is the one and only connection between the client GUI and the service context. The items of Attribute Name List is dynamically loaded from an XML Schema file. It is implemented by calling an XML handler to read the XSD file and returning all the required context information. However there are two problems.

First, what is the required context information? Or what does the GUI want the handler to get from this XSD file? Based on the analysis of Constraint Generator, for each leaf element, the name, data type and source (only for ontology type) is required. Take the movie example again,

```
<complexType name=”Movie” />
    <element name=”Title” type=”String”/>
    <element name=” Participator” >
            <element name=” Director” type=”String”/>
            <element name=” Cast” type=”bag” bagtype=”String”/>
    </element>
    <element name=” Catalogue” type=”ontology”
                                    source=”file:./Catalogue.owl”/>
</complexType>
```

From above complex “Movie” type, following information are harvested by the GUI:

Title, String

Director, String

Cast, bag,Sstring

Catalogue,ontology,file:./Catalogue.owl

Then comes the second problem which is how to store this information for future use? Unlike list or tree, Combo Boxes do not support extra data storage. The solution is to use a Vector to contain the information for each element.

Each object in this vector is an AttributeName instance. Here AttributeName is a Java bean which provides a data model for each record of required context information. The AttributeName bean has four fields: name, type, bagtype, and source. For example, the above Catalogue element would be converted into an AttributeName bean as:

```
AttributeNameInstance1{
    name=”Catalogue”;
    type=”ontology”;
    bagtype=null;        //the value would be null, if it is not defined in XSD
    source=” file:./Catalogue.owl”
}
```

Depending on these concerns, the process of loading all the service context into the client GUI could be broke down into the following steps:

1) XML handler using DOM to process the XSD file into a tree model and accesses every element
2) For each useful element, use DOM read its properties and use the set method of AttributeName bean to set the value of fields
3) Add the bean to the vector

4) Until all the useful elements have been added into vector, XML handler returns this vector to client GUI

5) Client GUI would load all the name fields of the beans in the vector into Attribute Name List and hold this vector for future use

6) When a user select an attribute from the combo box, client GUI would query the relevant bean in the vector and retrieve the data type and data source for the changing operator area and value area.

5.2.2 Namespace problem

As mentioned in 4.3.1.1, the “Catalogue” in “Programme” and in “Movie” shares the same name and the same OWL source, which might cause problems for KBN during the comparison.

It has been discussed in [8] that using a namespace to solve this problem is a good way to solve this problem and has already been applied to other applications.

However due to time constraints, the following was used to avoid the name confliction.

```
<complexType name=”Programme” />
    <element name=” Catalogue” type=”ontology”
                                    source=”file:./Catalogue.owl”/>
</complexType>
<complexType name=”Movie” />
    <element name=” Participator” >
        <element name=” Director” type=”String”/>
    </element>
    <element name=” Catalogue” type=”ontology”
                                    source=”file:./Catalogue.owl”/>
```

</complexType>

The Client GUI attaches the parent elements name as the prefix so that the”Programme.Catalogue“would be different from “”Movie.Catalogue”. Although they are using the same catalogue, they are still independent from each other. This is an adaptation of the namespace solution.

More over, the attach action would be executed as a loop until the head has been found. Such as the “Director” element in a Programme, the final attribute name loaded into attribute combo box would become “Movie.Participator.Director”.

Although this work around works well in this local project, it is not suggested for further develop. The namespace is definitely a better approach to solve name confliction problem.

Chapter 6 Evaluation

Because the project is to develop a client application, instead of testing the performance of KBN itself, the evaluation attempts to find out the usability of this client application. A usability test questionnaire is designed and tested with 10 volunteers. In addition, some relative evaluation and discussion will be included at the end of this chapter.

6.1 Test Design

6.1.1 Usability Test Questionnaire

The usability test questionnaire consists of four parts: Pre-Test, New Concepts, GUI Test, and Post-Test. Full version of this usability test questionnaire is attached as Appendix A.

6.1.1.1 Pre-Test

There are two objectives of the Pre-Test:

1) Collect users knowledge background about ontology and the KBN

   Because this Programme Guide is built on the KBN, the knowledge level of the ontology and KBN would influence the user activity.

2) Experience with similar IPTV application

   Some questions are designed to discover the users experience with some existed IPTV applications, including what applications they have used, what problem they have experienced and the users' personal preference when they are searching for TV programmes.

6.1.1.2 New Concepts

In this Personal IPTV Programme Guide, the way of describing personal interests with constraints is new to most users. Before using the Guide, the user needed to understand some new concepts, such as what a bag

value is.

The first object of this part of test was to introduce some necessary knowledge about ontologies and making subscription. Four concepts were introduced as follows:

a) Structure of Subscriptions and Constraints

b) Basic Operators

c) Bag Value and Bag Operators

d) Bag Type and Composite Relations

They are ordered from simple to complex. For each concept, it begins with a brief explanation and illustrated some examples. One or two questions are asked after learning for each concept.

The second object of this part, from the view of an investigator, is to find out how fast and how deeply users could accept these new concepts. Because a client application is designed for a user, it should not take too much time from a user for training. Once a user loses his/her patience, they lose the interest in using the application and in turn, the service provider loses the business. This issue would be recorded by conversation documentation during the test.

6.1.1.3 GUI Test

Based on the mastery of knowledge from the above training, the GUI Test attempts to

1) Show the different advantages of applying the KBN and ontology comparison with in different tasks

2) Test the interaction between the Users and GUI

This would be achieved by asking a user to use the GUI for the following five tasks. Each task is designed with a brief introduction, a simple scenario, and

some key clues for completing the task.

Task 1: Familiarization with the Client GUI

Full instruction was provided for this task so that the user just needs to follow the steps. It aimed to let the user apply the knowledge in their brain to real operation and understand the effect of each components in the GUI.

Task 2: Show the effect of the KBN ontology comparison

From task 2, the user needs to finish the task based on the clues with out detail instructions.

Task 3: Show the usability of the use of set value.

Task 4: By comparing the results with Task 3, show the effect of how constraints in a filter work in metadata match scheme.

Task 5: See the ability of KBN to keep updated

6.1.1.4 Post-Test

Post-Test question attempted to collect users feed back and comments based on their experience during the test. The experience included user feelings about how well they understood the new concepts introduced in 6.1.1.2 and the interaction with the GUI.

6.1.2 Time Stamp

During the test, the investigator attached a time stamp to record how much time the user spends on each section and each task. Because the time issue reflected the difficult level for a user to understand and accept new knowledge.

6.1.3 Face to Face

The test processes is a cooperative evaluation [5], in which both user and

investigator can ask each other questions throughout. All the conversation were recorded for analysis after, including questions, problems, as well as user behaviour explanations.

6.2 Test Results and Analysis

6.2.1 Volunteers Background:

The volunteers' knowledge background on the KBN and ontologies is presented in Table 1.

	KBN experts	UBC students, with limited ontology knowledge	Normal persons, with basic computer experience	Total Volunteers
Number	4	2	4	10

Table 1 Volunteers Knowledge Background

6.2.2 Discovery from Volunteers' Experience

6.2.2.1 Existed Tools

From the analysis of the Pre-Test questions, we can see that, more or less, people would search for TV programme or Videos on the internet. The tools they use can be classified into two kinds:

- Web Site search: such as Google, MSN, BBC News, entertainment.ie, YouTube
- Application: such as Joost, BitTorrent, iTunes

6.2.2.2 User Experience and Current Problems

Based on the experience of these Web Sites and other Applications, 50% felt satisfied that they could always find exactly what they searched for, and the other 50% felt not so satisfied that some times they will have some problems

in location a search. There are two main problems about the searching results included:

- Too many results returned that the user has to search within the list to try to find relevant result.
- The search results are not updated

Also, 20% think the reason they could not get expected answer is because the searching tool did not provide enough options for them to describe all their interests.

6.2.2.3 The Backend Rule of Query Results

For an application, the user should not care how that application in the background. But the fact is, especially for some search application, if the user knows the principle behind the GUI, he/she could query the expected result following the back-end rules so that a better quality of result returns. So, somehow, the way of how the application queries the results could affect the way that how user express their interests.

During the Pre-Test, 50% show they do not care how the backend rules of query results works, while another 40% prefer the Logical Match, such as the ontology comparison applied in the project. And this preference does not relate to the level of ontology experience.

Another important point is that most people, even for the users who do not care about the backend rules, think applying Logical Matching to a search is useful. This fact shows the ability of KBN ontology comparison has its usability.

6.2.2.4 User Preference and Problems

The last part of Pre-Test aimed to find out what features of a programme the user needed to select to express their interests. And these features should be included in the Metadata from the producer so that could be compared with the interests from users.

The results showed title and source quality are the main features users care for.

This could cause some future problems.

Firstly, the number of features is small. In the client application of the KBN, the user describes the interests with several constraints. More constraints in one subscription means the more specific the subscription is and the better quality results the KBN returns. But the fact is if the user is not aware of this feature of the KBN application and only specifies the title of the programme as used in other applications, the results might be worse than with other current applications. Because the order of results does not depend on the quality of the source, instead, it depends on the time of notification. It seams some mechanisms are still necessary for ordering the results, such as a priority weighting.

Secondly, who will guarantee the source quality? From the view of the users, the quality of the source is defined with the Metadata. If the source is provided by a “BBC” channel or the starRating is a large number, users might think the quality is good. But is it guaranteed that this programme is provided by a BBC channel? Therefore some authentications for checking Producer’s identification are required when attaching the Metadata.

6.2.3 Time Stamp Analysis

Time cost on section or task of each user is collected by calculating time stamp data.

6.2.3.1 New Concepts

Table 2 is the time result collected from New Concepts test. The table could be separated into three parts. All data is collected in minutes.

1) The data in the middle of the table is the average time unit that one type of user would spend on each section. For example a normal person would

2) The right column shows the average time spent on each section, without the concern of the knowledge background. Such as, on average, a person would spend 2.7 minutes to learning Basic Operators, including Number operators, String operators, and Ontology operators.

3) The last row at the button of the table is the average total time for one type of user to complete New Concepts learning. For example, a KBN expert would complete all sections within in 7.5 minutes while a normal person would need on average 15 minutes.

	Normal Person	UBC Students	KBN Experts	Average Time/Section
Subscription and Constraints	3	2.6	1.5	2.4
Basic Operators	3.5	2.6	2	2.7
Set Values and Set Operators	3.5	2	2	2.5
Set Type and Composite Relations	5	3.6	2	3.6
Total Time/Person	15	10.8	7.5	

Table 2 Time Stamp for New Concepts

Figure 18 is the histogram made from the right column. It clearly shows that the Bag Type and its Composite Relations would take more time to learn from a user. And the second section Basic Operators have taken a little more time than the others, probably it is because this section contains the understanding of ontology comparison.

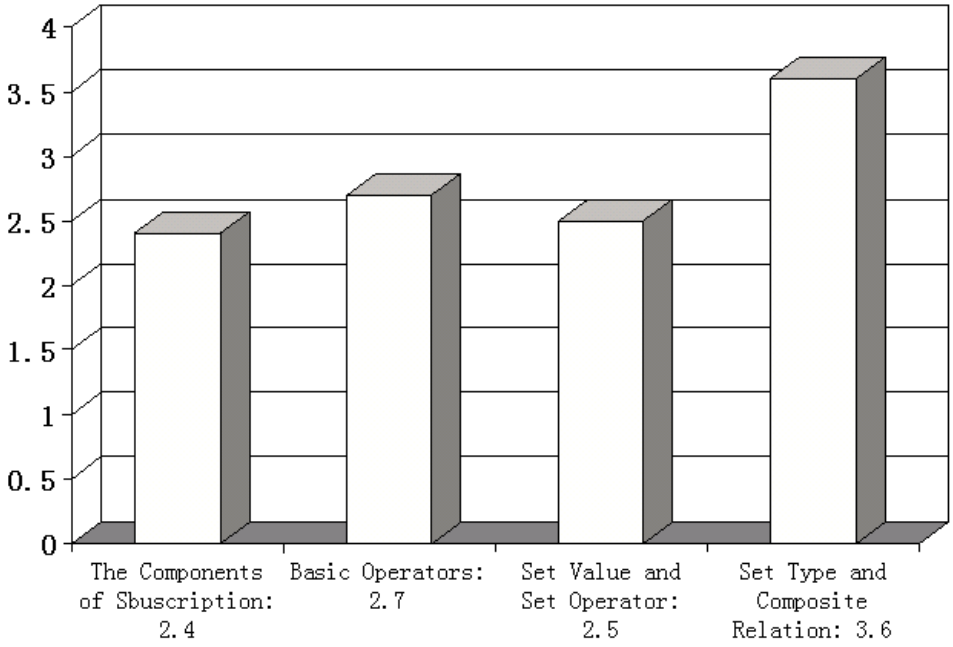

Figure 18 Average Time / Person / Section

Figure 19 is made from the button row of Table 2. The trend shows normal people nearly double the time cost if compared with a KBN expert. The reason could be ascribed to the learning phase of new concepts. A KBN expert might finish the questions without any difficulty, and a UBC student spend time on understanding the examples provided, while a normal person might not find the right answer even with the help from the investigator.

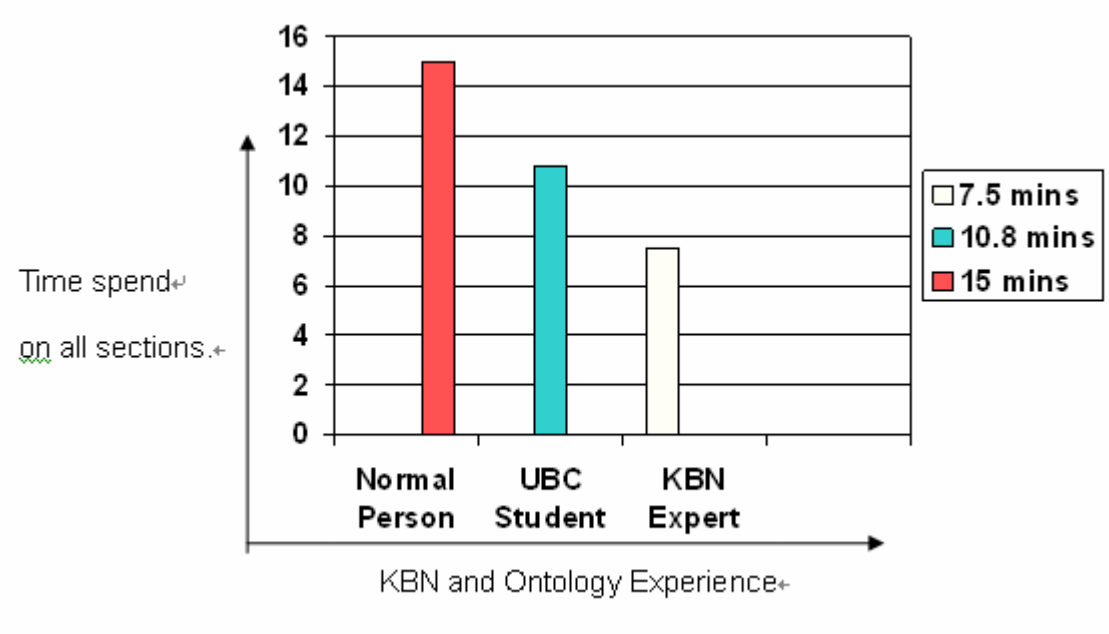

Figure 19 Total Time on All Sections for Different Kind of User

6.2.3.2 GUI test

Similar time issue is collected for GUI test, shown as Table 3.

	Normal Person	UBC Students	KBN Experts	Average Time/Task
Task1	6	5	2	4.3
Task2	6	5.2	6	5.7
Task3	4.5	3.3	4	3.9
Task4	3	2.6	1.5	2.3
Task5	3.6	2.6	3.5	3.2
Total Time/Person	23.1	18.7	17	

Table 3 Time Stamp for GUI test

Figure 20 presents the histogram of the right column of Table 3. Except task 2, task 1 has taken more time than other tasks, because it is the first time for the user to use this Programme GUI. And Task 2 takes 5.7 minutes on average, which is the highest value, because in task 2, users need to apply bag type and composite relations. During task 2, most users need help from the investigator, but in different degrees. Some need support for just bag operators or subbag operators, while some users totally lost. The investigator has to present the process of task 2 with a full explanation.

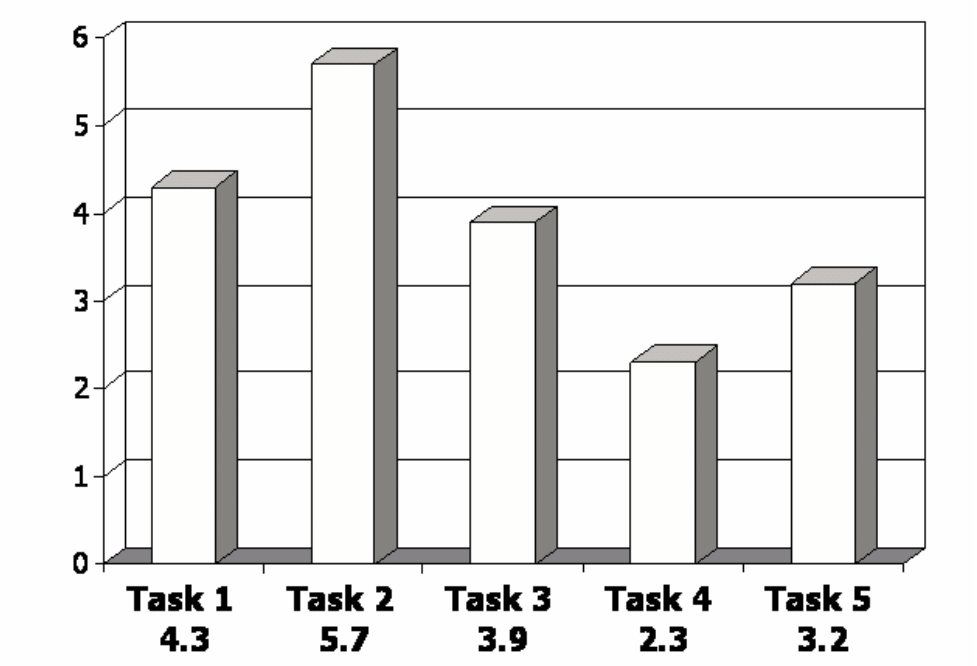

Figure 20 Average Time / Person / Task

Also, the total time for completing all tasks with different ontology knowledge backgrounds is compared in Figure21. Similar results the fact that with more KBN and ontology experience, less time is needed.

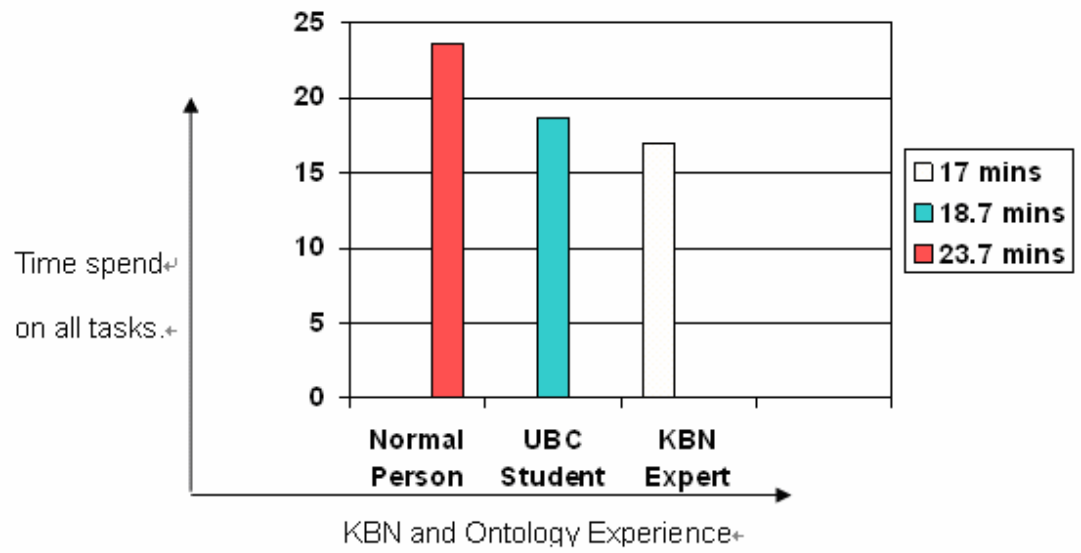

Figure 21 Total Time on All Tasks for Different Kind of User

From the analysis of the time stamp, a simple but clear conclusion can be made which is that the KBN and Ontology Experience of a user is an important factor, which influences the convenience of use on such client applications as for KBN.

6.3 User Feedback

Based on the answers from the 10 volunteers in the Post-Test and the observation of user behaviour during the test, a general user feedback is can be concluded as follows.

1) The most difficult: Operators, especially, Composite Operators

   When users the use client GUI to make a constraint, one of two cases often happens.

   Firstly, the user might forget to choose an operator, or only specify one operator for a composite relation. This behavior shows that most users do

not get used to express in a personal interests with operator(s). Fortunately, the missing operator(s) problem is avoided in advance with validation checks and Alter messages.

Secondly, most users would select the “equals” relation as their first choice. The conversations with user are recorded for analyzing the reason of these “equals” selection. The result shows this is because of the “is” expression in users’ mind. When a user attempts to search for something, the thinking in their mind is always “what I am looking for is ***”. This is because lots of search engines, such as Google, only requires users to specify the value of the search object without any operator concept.

As a result, how to ease and simplify the use of operators in the client application for the KBN is very important. Especially for the use of composite relations, because lots of users complained about these two operators.

2) The way to describe interests with constraints: 70% vs. 30%

Although a user might find it difficult to make constraints, there are still 70% of users who enjoy using this way to describe their interests. This is because they think the constraint enable them to express their interests more personally.

3) Operators and value types are enough to express interests: 90% vs. 10%

90% of users feel the four operator sets and three types of value with single or bag options are enough, more than enough, for them to describe their interests about a programme. The cause of 10% who are not satisfied might be the a problem with his/her knowledge level. An explained could be that they do not know how to make constraints with the operators and value types in some complicate cases.

4) Feel useful to apply ontology comparison into these Client Applications: 70% vs.30%

After showing the advantage of the KBN and the ontology comparison

with five tasks in GUI test, 70% feel it is helpful to improve the current problems with existed IPTV applications or some other search engines. Such as, the updated problem, and huge number of results problem. But there are 30% who do not agree with this advantage. They agree that the quality of results has been improved, but the improvement is not enough to attract them to use such a complicate application.

5) The relation between knowledge level and satisfied level

   This issue has already been discovered from the result of the time stamp. 75% of who are not so satisfied with the application are normal persons and feel too complicated to use constraints.

6.4 User comments

Some comments and suggestions from users are collected and organized as bellow. Because the test focused on the client GUI and hid the back application architecture, most comments and suggests are about the improvement of the Client GUI.

1) Operators should be further translated into more nature language, such as using “contains” to replace “has sub string” as a String operator

2) A Subscription List should be provided to enable the editing of submitted Subscriptions. Actually, this has been considered in the implementation phase, but because of time restraints, it has not been finally achieved.

3) Tool Tips, Tool bars and Label Indicator would be helpful. The test is a face-to-face test so that if there is any question, the user could ask the investigator for help. But if the application goes to the real world, some help files would be necessary.

4) Instead of displaying the results in the order of arrive, they should be grouped with relevant subscription. And, as discussed in design phase, the name of the subscription should support user defined names which contains real meaning and ease the user to manage their own search items.

5) Case Sensitive support. Although the KBN is case sensitive, this problem should be solved before sending the subscription to KBN.

6) Providing suggested search terms could improve the efficiency of use by speeding up the interaction between user and application [5]. When a user enters “Brad”, “Brad Pitt” should be displayed for selection. Lots of applications and search engines have already achieved this and enhanced their efficiency. The search terms could include two kinds of selections, something which is popular within society and something a user has already imported several times, similar to the history record of a web browser.

6.5 Relative Evaluation and Discusses

Because of time issues, technical difficulties, some functions or designs have not been fully implemented and some interesting topics and discussions have not been further researched. As a relative evaluation, aiming to leave some future direction for somebody who might have his interest in further developing this client application, some of the main issues are presented and discussed below:

1) Further improve the client GUI as discussed in 6.4 to simplify the use of KBN.

2) Apply real namespaces into the XSD file as discussed in 5.2.2 so that the scalability would be enhanced.

3) Support semantic mapping.

   Because the emphasis of this project is on the client side, there is not a lot of research for the producer side. But during the design phase to keep the consistence between producer and consumer, there is a problem with multi producers.

   As mentioned in the chapter one, one of the reasons to choose IPTV as an instance for development is because there is no existing application

that integrates all the available online TV channels. To be more specific, the design of the XSD file only keeps the consistency between the KBN the input and output, but does not promise to keep the consistency between different producers. As those channels are already existed, it is impossible to ask them to change their own ontology or metadata structure.

One approach is semantic mapping [6]. In order to enable the metadata to match the schema within the KBN, there will be a need to map between multiple channels at inter-organization level.

4) The updated ability could be advantage as well as disadvantage.

During the GUI test in task 5, an NBA live game example is used to present one of the advantages of applying the KBN to the IPTV Programme Guide which is that it allows a user to always keep information updated and filter out all old information.

This is because the KBN would only compare the Subscription with those Notifications, which are sent after that Subscription. This could be an advantage as well as a disadvantage in this IPTV Programme Guide.

Case 1: Notification sent after the Subscription, which means the programme has not begun and the user will get the time and other relevant information. This is what the application wants.

Case 2: Notification sent before Subscription and the programme is ongoing. Advantage or disadvantage depends on the feature of the programme and user preference. For a live game, such as NBA or World Cup, it is advantage. For a movie, some users do not care missing the start of the movie, while some users care so much about the integrity.

Case 3: Notification sent before Subscription and the programme has been finished. It could be disadvantage if the content of the programme is still valuable even it has finished, such as weather information for next week.

One approach is to make KBN cache valuable information for a certain time, but one problem is how long the information should be cached. Another problem is the cache ability would reduce the capability to support distribute service.

A second approach could be asking the producer to repeat sending the notifications. Similarly, the problem is how long should a producer keeps repeating and at what frequency?

Above are some issues that might be valuable and interesting for further work.

Chapter 7 Conclusions and Further Work

This chapter aims to conclude the main contributions from this implementation of this Personal IPTV Programme Guide. Some further work is discusses at the end.

7.1 Conclusions

Lots of tests have be done to evaluate the performance of the KBN and lots of the KBN's powerful ability have been discussed in the fields of semantic web service, distributed systems, automated interoperation information processing[2], autonomous networks and so on. But the KBN has not been applied to real systems, and there is no client application that takes advantage of the ontological comparison and bag comparator matching algorithm supported by KBN.

One of the major contributions of this project is to develop and implement such a client application for the KBN, which is called the Personal IPTV Programme Guide. This guide attempts to help a user find his/her interested TV programme among thousands of available programmes from different online channels. As the first step for the KBN to move from lab to public usage, the most difficult work is to let the public users, who do not have any

experience with KBN and ontology knowledge, to understand and accept these new kinds of client applications and finally enjoy the benefits provided by KBN.

Another major contribution is the usability test of this Personal IPTV Programme Guide. From the answers and user feedbacks, lots of valuable issues discovered from a view of user preference, including comments on user interface and some suggestions for future work.

The last major contribution is the architecture design of the application to use an XML schema file (XSD file) separates the client GUI and semantic service context. The support on independency between user interface and the service context eases the matainance and update for an individual client application and enables the extensibility and flexibility between different client applications for the KBN.

Based on the experience of research, design, implementation and the analysis of usability test, a simple conclusion could be made that an affirmation should be awarded to the ability of KBN and its ontology comparison, but to become a real client application, lots of works is still required to simplify the user operation.

7.2 Future Work

The future work could include two parts: improvement and further development.

Based on user comments and problems discovered from design, implementation and testing, the client GUI should be redesigned to ease the use of such personal guide, and some technical mechanism should be applied into the application to enhance the scalability, such as semantic mapping and namespace usage.

The further development could be pulling real time results from the internet and feeding into the system instead of using simulated TV metadata. And the final object is to integrate this client GUI into real system. For this Personal

IPTV Programme Guide, the potential system integration plan includes:

1) Overlap KBN with MPLS network so that once the KBN discovers the source for a user, the TV stream could be routed fast over MPLS and directly from Producer to Consumer based on its content

2) Allow the user to request a source TV stream with SIP (Session Initiation Protocol)

3) Use semantic mapping to convert individual TV stream metadata from different channels into valid metadata structure to support KBN comparison.

And a preview of application architecture is shown as Figure 22

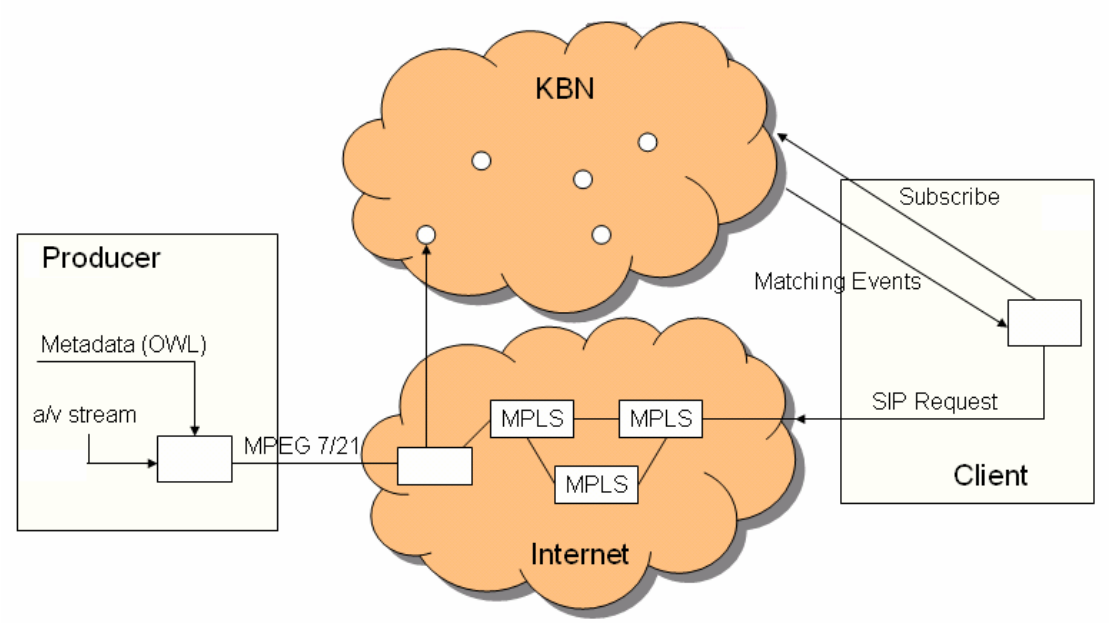

Figure 22 System Architecture of an Entire Personal Online IPTV Application

Bibliography

[1] John Keeney, Dave Lewis, Declan O'Sullivan, *Ontological Semantics for Distributing Contextual Knowledge in Highly Distributed Autonomic Systems*

[2] http://en.wikipedia.org/wiki/Publish/subscribe

[3] Dominik Roblek, 2006, *Decentralized Discovery and Execution for Composite Semantic Web Services*

[4] Nielsen and Molich, 1990, *Heuristic Evaluation of User Interfaces*

[5] http://en.wikipedia.org/wiki/XML_Schema

[6] Information Architecture and Knowledge notes, UBC 2006-2007

[7] Hameed et al. 2004

[8] Dave Lewis, *KBN Naming Convention,*2007

[9] http://www.cucirca.com/2007/02/21/13-places-to-watch-tv-online-for-free/

[10] http://www.tv-links.co.uk/

[11] http://www.ppstream.com/

[12] http://en.wikipedia.org/wiki/Ontology_%28computer_science%29

[13] http://en.wikipedia.org/wiki/Web_Ontology_Language

[14] http://www.chiariglione.org/mpeg/standards/mpeg-7/mpeg-7.htm

[15] http://dynamic.abc.go.com/streaming/landing

[16] http://en.wikipedia.org/wiki/RSS

[17] http://en.wikipedia.org/wiki/Java_Swing

[18] Brian Cole, Robert Eckstein, James Elliott, Marc Loy, David Wood, 2002, *Java Swing*, O'Reilly publisher

[19] http://jena.sourceforge.net/

[20] http://java.sun.com/j2se/1.4.2/docs/api/org/w3c/dom/package-summary.html

[21] http://www.roseindia.net/xml/dom/

[22] http://en.wikipedia.org/wiki/Document_Object_Model

Appendix A: Usability Test Questionnaire

Pre-Test Questionnaire

Q1. Do you have experiences in searching TV programmes or news on Internet?

A. Always

B. Very often

C. Some times

D. Not often

E. Never

Q2. How satisfied you feel with the application you have used?

A. I always find exactly what I want.

B. Some times it works, some times not.

C. Unsatisfied.

Please indicate any applications you have used:

Q3. Have you experienced any of the problems below? (Multi Choices)

A. Too many results returned that I have to search within the list again.

B. The result is not updated.

C. I cannot describe all my interests, because of the application limits.

Please indicate any other problems you have experienced:

Q4. Which way you prefer when you search something?

A. String match

B. Logical match (If you want information about Ireland, and the information is from Dublin, then you should get this information)

C. I do not care

Q5. How much do you know about Ontology and Ontology Comparison?

A. Expert

B. Moderate

C. Little

D. None

Q6. How much do you know about KBN?

A. Expert

B. Moderate

C. Little

D. None

Q7. Please briefly describe what you will do if you want to watch “Spider Man 3”? (Including what application and want you will type in for search)

Step 1.

Step 2.

Step 3.

Q8. What do you look for when you search for a video programm?

Client Application for KBN Overview (IPTV simulation)

The main issue of this IPTV simulation, which could be an instance of Client Applications for KBN, is to apply the ontology comparison into the routing of video streams. It allows the user to fully describe his/her interests, and then the interests will be translated into Subscription when published into KBN. KBN will return all the events that 100% matches the Subscription.

Making Subscriptions

Because the only thing a user needs to do is to describe the interests, it is necessary for the user to know how to make a subscription. There are several concepts that might be new to new users.

The Components of Subscription

One subscription can contain several constrains, and each constrain consists of Name, Operator, and Value.

An example:

Subscription 1.

Constrain 1: Location = “Dublin”

Constrain 2: Channel = “BBC2”

Constrain 3: Catalogue = “Sports”

Based on Subscription 1, you could get all the Sports events happen in Dublin from BBC 2 channel.

Q1. Do you understand the structure of Subscription and Constrains?

A. Yes

B. No

If you have any questions, please ask the investigator:

Basic Operators

The operator in one constrain define the relationship between the user expected value and the user defined value of one attribute. The reason is that the user might not know exactly what the value is, or expect the value in a certain range. There are three different sets of operators for different types of value: Number type, String type, and Ontology type. Detail operator definitions are included in Appendix B.

An example:

Subscription 2

Constrain 1: Rating-star >3 // Number type

Constrain 2: Channel *has prefixed* "BBC" // String type

Subscription 2 means the results should have the rating-star greater than 3 and the source must be a BBC channel.

Another Example;

Location has an ontology type value.

Constrain 1: Location *more specific* Dublin

Constrain 2: Location *less specific* Dublin

The difference is that, if the source metadata is Location is Ireland, Constrain 1 will stop it, but Constrain 2 will allow it to pass, because Ireland is a super class of Dublin, which means Ireland is less specific than Dublin.

A reminder is that, if the source metadata is Location is Dublin, both of them will pass it. Because both "*more specific"* and "*less specific"* include the "equals" relationship.

"*more specific"* means more specific than or equals to.

"less specific" means less specific than or equals to.

Part of Catalogue. owl tree structure is:

- Catalogue
  - …
  - Sports
    - Basket Ball
      - NBA
      - WNBA
    - Foot Ball
      - …
  - …

Q1. Please circle all the results that you think satisfied this constrain (see Appendix A for operator definitions):

1) Constrain 1: Catalogue *equals ontology class* “Basket Ball”
   A. Catalogue *equals ontology class* Basket Ball
   B. Catalogue *equals ontology class* Foot Ball
   C. Catalogue *equals ontology class* NBA
   D. Catalogue *equals ontology class* WNBA

2) Constrain 2: Catalogue *more specific than* “Basket Ball”
   A. Catalogue *equals ontology class* Basket Ball
   B. Catalogue *equals ontology class* Foot Ball
   C. Catalogue *equals ontology class* NBA
   D. Catalogue *equals ontology class* WNBA

Q2. Do you understand the meaning of these basic operators?

A. Yes

B. No

If you have any questions, please ask the investigator:

Set Values and Set Operators

Instead of a single value, the user can use a Set value to describe the domain of the expected value. For example, it is reasonable to describe the “actors” attribute of a programme as a set value, cause there always be more than one actor.

An example:

Constrain 1: actors *is super set of* [Brad Pitt, Jennifer Aniston]

This constrains means the result must contain at least both Brad Pitt and Jennifer Aniston

Q1. Do you understand what Set value is?

A. Yes

B. No

If you have any questions, please ask the investigator:

Q2. Do you understand the meaning of Set operators?

A. Yes

B. No

If you have any questions, please ask the investigator:

Set Type and Composite Relations

The type of Set value depends on the element type. If the elements are String type, the Set is a String Set. For a certain Set, the relationship between the elements in two Sets can be described using a SubSet operator.

An example:

For an Int Set, [9, 8,7] *is subset of* / < [10, 11, 12], because all the values are smaller than the number in the second set.

But [9, 8, 7] *is subset of* / < [9, 9, 9] is wrong, because 9 is not less than 9.

Q1. Do you understand what Set type is?

A. Yes

B. No

If you have any questions, please ask the investigator:

Q2. Which of following you think is the right expression of ExpectedWord is a super String Set with all elements contain ”or”:

A. ExpectedWord *is sub set of* / *subString* [“or”]

B. ExpectedWord is super set of / *subString* [“or”]

C. ExpectedWord *is super set of* / *has prefixed* [“or”]

D. ExpectedWord *is sub set of* / *not equals* [“or”]

Q3. The use of combination of Set operator and SubSet operator is called Composite relationship.

Do you understand what Composite relationship is?

A. Yes

B. No

If you have any questions, please ask the investigator:

GUI Test

Task 1

Aim: Getting familiar with the Client GUI

Scenario:

You have a child in school age, and you want to find some suitable English TV programme for him.

1) Language is English
2) Catalogue is school age kids programme.
3) Because the programme is for your child, you want the programmes with good quality, so that the star-rating should be high, at least 3 star.

Instructions:

1) Language is English
   a) Choose “Programme.Language” from the Attribute Name List
   b) Operator “equals”
   c) Type value in the value text area “english”
   d) Press “Add” button to add this constrain
2) Catalogue is school age kids programme.
   a) Choose “Programm.viedo,Catalogue” from the Attribute Name List
   b) Operator “*More Specific*”
   c) See the Tree structure on the right.

      Select “Programme” -> “Kids”-> “School Age”
   d) Press “Add” button to add this constrain

3) Because the programme is for your child, you want the programmes with good quality, so that the star-rating should be high, at least 3 star.
    a) Choose “Programm.starRating” from the Attribute Name List
    b) Operator “>=”
    c) Type in value “3”
    d) Press “Add” button to add this constrain
4) Press “Submit” button to submit this subscription
5) Check the result in the right Result List

Task 2

Aim: See the effect of KBN otology comparison

Scenario:

You are going to travel in Spain and France, so you want to know what are for the next week.

1) You want Channel is any BBC channel
2) You want all the Weather News
3) You need to specify the country as Spain and France

Task 3

Aim: There are two aims for Task 3 and Task 4.

First, from task 3, you would see the usability of the use of set value.

Second, by comparing the results with Task 4, you would see the effect of how filter (a subscription with contained constrains) works for metadata match scheme.

Scenario:

You are a member of a Movie Club. Next week, each member needs to recommend an exciting movie. You have a film in mind, but you can not remember the name. You know Brad Pitt and Angelina Jolie has participate the casting. In order to find out this movie based on a small piece of information, you will

1) Make Brad Pitt and Angelina Jolie as the value of actors.

Task 4

Aim: By comparing the results with Task 4, see the effect of how filter (a subscription with contained constrains) works for metadata match scheme.

Scenario:

After doing Task 3, you will see lots of information, in which most are the gossip of Brad Pitt and Angelina Jolie. So you do not want to waste time to find the movie, you will detail your subscription.

1) Make Brad Pitt and Angelina Jolie as the value of actors.
2) Specify the catalogue as Movie

Task 5

Aim: See the ability of KBN to keep updated, by comparing the results from the same subscription but submit at different time,

Scenario:

You are a fan of NBA games. Now, Rockets is playing with Jazz. The match is so exciting that the scores changes every second. You start the Client GUI to search for the game:

1) Make “Rockets” and “Jazz” as the set value of team.
2) Specify the catalogue as NBA

Post-Test

Q1. Do you feel difficult to use the application during the tasks?

	Easy	Little Difficult	Difficult
A. Use the Attribute List			
B. Use Basic Operators			
C. Use Set Operators			
D. Use SubSet/ Set Type Operators			
E. Use Set Value			
F. Understand Ontology Comparison			
G. Make a constrain			
H. Make a subscription			
I. Check the results			

Any other difficulties?

Q2. Do you like the way you describe your interests during the tasks?

A. Yes

B. No

Q3. Do you think the operators and value types are enough for you to express your interests?

A. Yes

B. No

Q4. Do you think applying ontology comparison in these kinds of client application is useful?

A. Yes, very useful.

B. Useful, but still need to be improved

C. No, even worse than the existed application.

Q5. Comments and Suggestions:

Appendix B: Operators

Number Type Operators

"=" EQUAL

"<" LESS THAN

">" GREATER THAN

">=" GREATER OR EQUAL

"<=" LESS OR EQUAL

String Type Operators

"=" EQUAL

">*" HAS PREFIX e.g., "software" >* "soft"

"*<" HAS SUFFIX e.g., "software" *< "ware"

"!=" NOT EQUAL

"*" SUBSTRING e.g., "software" SS "war"

Ontology Operator

"@=" EQUIVAL

"@>" MORESPEC e.g. Apple is more specific than Fruit

Apple is more specific than Apple

"@<" LESSSPEC e.g. Fruit is less specific than Apple

Fruit is less specific than Fruit

Set Operator

"#=" EQUAL SET e.g. [1,2,3] equals to [1,2,3]

"#>" SUPER SET e.g. [1,2,3] is a super Set of [1,2]

"#<" SUB SET e.g. [1,2] is a sub Set of [1,2,3]

Set Operator with SubSet Operator

Set Operator describes the relationship between Sets.

SubSet Operator, which depends on the type of the Set, describes the relationship between the elements in the Sets.

e.g. [1,2] #< < [1,2,3]

[1,2,3] #> > [0,1,1]

The following expression is wrong:

[1,1,1] #< < [1,2,3] cause 1<2,1<3, but the last 1 not less than 1

[1,2,3] #> < [2,3,4,5] cause [1,2,3] is not a super Set of [2,3,4,5]

[[1],[2],[3]] #= = [1,2,3] cause the type of Set is different