Client Service Capability Matching

A Thesis submitted for the Degree of Master of Science

by

Oliver Martin Lyttleton, B.Sc Computer Applications

Student no 96300493

School of Computing,

Dublin City University

July 2004

Supervisor Dr David Sinclair

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of M.Sc in Computer Applications is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed: Oliver Lyttleton

(Candidate) ID No.: 96300413

Date: 25-09-04

Acknowledgements

I would like to thank my supervisors David Sinclair and David Treacy for their help, advice, and encouragement during the last two years

I also would like to thank Sun Microsystems and Enterprise Ireland for sponsoring this work

Finally I would like to thank my family for their support during my time in DCU

Table of Contents

Abstract

In order to tailor web-content to the requirements of a device, it is necessary to access information about the attributes of both the device and the web content Profiles containing such information from heterogeneous sources may use many different terms to represent the same concept (eg Resolution/Screen_Res/Res) This can present problems for applications which try to interpret the semantics of these terms

In this thesis, we present an architecture which, when given profiles describing a device and web service, can identify terms that are present in an ontology of recognised terms in the domain of device capabilities and web service requirements The architecture can semi-automatically identify unknown terms by combining the results of several schema-matching applications The ontology can be expanded based on end-user's interaction with the semi-automatic matchers and thus over time the application's ontology will grow to include previously unknown terms

1. Introduction

The Internet is claimed to be a global medium, information can be accessed anytime, anywhere Yet our ability to access information, and the quality of our experience, depends on the capabilities of the device we use to access the information

There are an increasing range of devices used to connect to the Internet, PCs, mobile phones, PDAs, set-top boxes, to name but a few All these devices have varying capabilities, yet most web content is designed with the assumption that it will be accessed using a standard PC As the range of devices with Internet connectivity grows, the assumption that web-content designed solely for PCs is universally accessible becomes increasingly naive

The W3C Device Independence activity [1] seeks to avoid the fragmentation of the web into spaces that are only accessible from subsets of devices

The list below illustrates the variety of devices that can access the web, and their differing capabilities

Workstations (eg desktops, laptops)

Characteristics Powerful processors, large displays, audio capabilities, large amounts of memory, persistent storage Input capabilities include keyboard, mouse/touch pad Can use a range of different network connections, both wired and wireless

Personal Digital Assistants (PDA)

Characteristics Physically smaller than workstations Less powerful processors, less memory and less persistent storage than workstations Input capabilities are also more limited (eg stylus and a writing surface on the PDA, or a miniature keyboard) Highly portable

Mobile Phones

Characteristics Small in size, highly portable Have a need for extended battery life, therefore use lower power processors and less memory than PDAs Small screens, numeric keypad for input Network connection available via phone's link to its network, and is typically slower than those available on Workstations

Voice Systems

Characteristics Provide connection to the web from standard telephone handsets No display, output is audio only Input via voice recognition, or numeric keypad

Interactive Television Systems

Characteristics Lower resolution than Workstation Input limited by remote control

It is clear that there are many challenges for authors who wish to create device-independent web-content For example, screen size and resolution are particularly important issues to consider They are crucial in determining the physical layout of web content Authors may need to design different physical layouts and different ways of organising web content in order to take into account the differences in size and resolution of the displays in use The differences in display size range from potentially quite large (workstations) to miniscule (mobile phones) and can even be non-existent (voice systems) There is a similar range of variety in the display resolution that devices can support

Input capabilities can also vary enormously Workstation devices with full keyboards can be easily used to input large quantities of data, whereas it is not as easy to do so using the keypad of a mobile phone The ease of use of a device's input facilities is an important consideration for web designers The interaction between a web application and a user may need to be simplified for use on devices with limited input capabilities Certain functions may even have to be omitted when using such a device It may not be viable to attempt a complex registration procedure involving large numbers of forms to be filled out using the keypad on a mobile phone

It is important that when authoring web content, designers are able to ascertain the input capabilities of devices used to access their content, in order to define how users can access web content appropriate for their access device

The speed of the network connection available to a device is also an important consideration when considering what types of web content are suitable for that device Large images or video clips may be provided in lower-quality formats suitable for lower connection speeds Content providers may even offer text alternatives for use on very slow connections

The W3C Device Independence activity [1] recommends that web-content be tailored according to the properties of the access device In order to tailor web-content for a particular access device, it is necessary for personalisation applications to access information detailing the attributes of both the access mechanism and the web content

For example, technologies such as XML [21] and XSLT [32] can be used to further device independence Data can be described in a platform neutral way by using XML XSLT can be used to transform XML documents into other XML documents A WAP-enabled mobile phone can use XSLT to transform an XHTML document into a WML document

The Authoring Challenges for Device Independence note [33] defines Media Resource Selection as Device Independence Authoring Challenge 3 15

> "Authoring techniques that support Device Independence should provide the ability to select an appropriate resource from the alternatives available according to the capabilities of the device"

For example, consider a device with a slow network connection If this device is used to access web content that contains high quality video requiring a high bandwidth connection to view, lower quality video which is more appropriate for a slow network connection should be offered as an alternative by the web content provider

Device Independence Authoring Challenge 3 7 is defined as

> "Authoring techniques that support Device Independence should support the use of different versions and types of media, such as images and audio clips, on different devices with different delivery contexts "

For example, rather than a video clip, an audio clip might be appropriate on a device with limited display capabilities, such as a mobile phone When a mobile phone is used to access a video clip, the provider of the clip may offer the audio clip instead

In each of these scenarios, the web content provider must be able to access information about the device used to access the web content (markup languages supported, network connection speed, display capabilities)

In particular, the W3C Device Independence activity focuses on methods by which the characteristics of the device are made available for use in the processing associated with Device Independence[34]

The Authoring Challenges for Device Independence note[33] defines as Device Independence Authoring Challenges 4 2, Capability Abstraction

"Authoring techniques that support DI should provide mechanisms that allow authors to express the user experience they wish to achieve using abstract representations of the underlying capabilities of the device "

Personalisation of web content can be carried out automatically by applications which can use the semantics of terms found in these abstract representations (or profiles) to determine what transformations (if any) are required to enable web content to be accessed on a particular device The set of terms that are used by someone to describe devices can be considered to be an ontology

An ontology defines the terms representing the concepts that are assumed to exist in a domain of knowledge (eg computer devices, finance, real estate, etc) Ontologies are used by humans and computer applications that need to share domain information [2], [3] Ontologies include machine-readable definitions of concepts in the domain and the relationships that hold between them Ontologies are a key concept in the emerging Semantic Web [4], and are a way of representing the semantics of web resources and enabling the semantics to be used by web applications and intelligent agents

There are many ontologies representing devices and web content from a variety of sources in existence, and their number is growing If these ontologies continue to proliferate, interpreting profiles from all these sources will be difficult to automate If there is no standardised ontology representing the domains of device capabilities and web content, personalisation applications will frequently encounter terms in profiles that are not in their own ontologies If a personalisation application can't understand the meaning of a device/web content profile, it can't determine what changes (if any) need to be made to the web content

In this paper we will describe an architecture called the "Client Service Capability Matcher" It identifies element and attribute names in an RDF [5] profile which are present in its own ontology (the "rules base") A rules base is a set of rules indicating that a term occurring in one data source is semantically equivalent to another term appearing in another data source The Client Service Capability Matcher implements the rules base as a table in a MySQL database The table contains two columns TERM and SYNONYM

Below ıs a sectıon of thıs rules base

Table 1 1 sectıon of rules base

TERM	SYNONYM
MB	Megabytes
MB	Meg
MB	Megs
Soundcard	Soundboard
NetworkConnectıon	ConnectıonSpeed
Mıcrophone	Mıc
Mıcrophone	Mouthpıece
Speakers	Loudspeakers
Headphones	Phones
Html	Htm
Txt	Text
Txt	Plaıntext

When processıng profiles descrıbıng a devıce, the rules base can be used to determıne the semantıcs of the terms used ın the profile

Below is a sample profile describing a device

Figure 1 1 sample device profile

```
<deviceProfile>

        <mem>128 MB</mem>

        <screenRes>1024 x 768</screenRes>

        <soundCard>Soundblaster 64</soundCard>




</deviceProfile>
```

The rules base can be queried using SQL [25] statements, in order to determine if the element names in this profile are contained in the Client Service Capability Matcher's ontology, for example

> **SELECT TERM FROM RULES_BASE WHERE SYNONYM='mem'**
>
> This statement will return the value 'RAM'
>
> **SELECT TERM FROM RULES_BASE WHERE SYNONYM='screenRes'**
>
> This statement will return the value 'Resolution'
>
> **SELECT TERM FROM RULES_BASE WHERE SYNONYM='soundCard'**
>
> This statement will return the value 'Sound'

The results from these SQL statements indicate that the terms "mem", "screenRes", and "soundCard" are semantically equivalent to the terms "RAM", "Resolution", and "Sound" respectively in the Client-Server Capability Matcher's ontology

The Rules base can determine the semantics of terms used in profiles, even when different terms are used to represent the same thing. The Client Service Capability Matcher can process profiles from multiple sources which do not use the same ontology.

However, sometimes the rules base will not return a semantically equivalent term when presented with a particular query. This happens when the query is composed using a query that does not appear in the SYNONYM column of the database table representing the rules base.

If carried out manually, finding semantically equivalent terms between two schemas can be a time-consuming, tedious effort which becomes increasingly impractical as the size and the number of the schemas increases. In this situation, the personalisation application needs to use a probabilistic method of determining what term from its own ontology the unknown term is most likely to represent.

Schema matchers are applications that use heuristic algorithms to provide suggestions for semantically equivalent terms between schemas. The ontology composed of the terms in the Client Service Capability Matcher's rules base and the ontology composed of the terms used in a profile describing a device or web content can be considered as schemas.. A schema matcher can suggest possible semantic mappings between terms in different schemas to a human, who can then accept or reject this semantic mapping. These applications are called semi-automatic schema matchers, because they still require human intervention to match semantically equivalent terms. These schema matchers use machine learning techniques to create semantic mappings. For example, the Naïve Bayes algorithm is a text classification algorithm whose effectiveness has been proven in a variety of applications. Spam filters [18] and Natural Language Processing applications [19] have all used this algorithm with a degree of success.

If an unknown term is encountered by the Client Service Capability Matcher, the results of three semi-automatic schema matching applications are combined in order to determine what term in the rules base the unknown term is most likely to be semantically equivalent to. The rules base can be expanded to include new semantic mappings between terms, based on the results returned by the user's interaction with the semi-automatic matchers Thus, as new terms are encountered, their meaning can be ascertained and eventually the terms can be added to the application's rules base

The structure of the remainder of this thesis is as follows In chapter 2, we analyse some existing approaches to matching semantically equivalent terms from heterogeneous ontologies We highlight the strengths and weaknesses of these approaches We outline what we desire from a schema matching application in the context of the particular problem area we are working in

Chapter 3 is an overview of the architecture of our system Each component of the system and the interaction between these components is described here

Chapter 4 is a description of the implementation of this architecture This contains details of the algorithms employed by the system's components, and the APIs and program libraries that were used in their implementation

In chapter 5 we evaluate the effectiveness of the system's architecture and the efficiency of its implementation

In chapter 6 we describe the data used to test the system, and analyse the results obtained

Chapter 7 presents our conclusions

2 Mediating between heterogeneous data sources

2.1 Creating semantic mappings between terms from different ontologies

A schema is a representation of the structure of data in a database. A schema can be represented graphically (eg as a graph using nodes and edges) or textually (using XML). Schemas are used to define the structure of information used by an application.

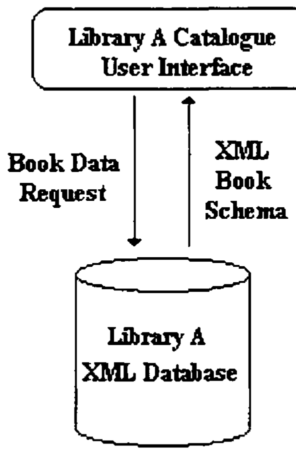
Here is an example of an XML schema representing a book:

Figure 2.1: Library A schema describing the book "Compilers: Principles, Techniques, and Tools"

```
<BOOK>
        <AUTHOR> Ullman, Jonathon </AUTHOR>
        <TITLE> Compilers: Principles, Techniques, and Tools </TITLE>
        <PUBLISHER> Addison-Wesley </PUBLISHER>
        <YEAR> 1985 </YEAR>
</BOOK>
```

The schema in figure 2.1 describes a book. This schema might represent the structure of a table in a database used by a library (Library A) to keep track of what books the library currently has (see figure 2.2).

Figure 2 2 Interaction between library database and user interface



The schema representing a book in another library's database (Library B) may be different from the Schema in figure 2 1 however It could look like this

Figure 2 3 Library B schema for book "Compilers Principles, Techniques, and Tools"

```
<BOOK>

    <AUTHORNAME>

        <SURNAME> Ullman </SURNAME>

        <FIRSTNAME> Jonathon </FIRSTNAME>

    </AUTHORNAME>

    <TITLE> Compilers Principles, Techniques, and Tools </TITLE>

    <PUBLISHED_BY> Addison-Wesley </PUBLISHED_BY>

    <YEAR_OF_PUBLICATION> 1985 </YEAR_OF_PUBLICATION>

</BOOK>
```

The schema in figure 2 3 is different in two ways from the schema in figure 2 1 its structure is different, and it uses a different vocabulary In figure 2 1, the value of the "AUTHOR" element contains the surname, followed by a comma, followed by the first name of the author

<AUTHOR> Ullman, Jonathon </AUTHOR>

However, this information is structured differently in figure 2 3 The "AUTHOR" element contains two subelements, "SURNAME" and "FIRSTNAME"

```
<AUTHORNAME>
    <SURNAME> Ullman </SURNAME>
    <FIRSTNAME> Jonathon </FIRSTNAME>
</AUTHORNAME>
```

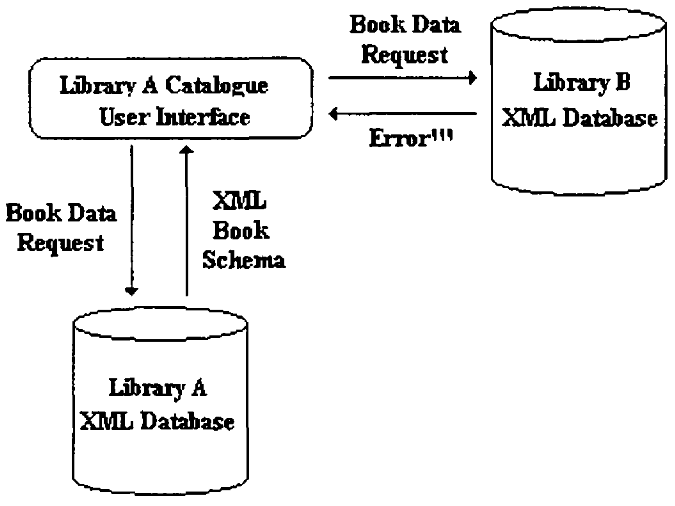
An ontology is a list of all the concepts that are assumed to exist in a particular domain of discourse It is a formal specification of how to represent the objects and concepts that exist in a particular area and the relationships that hold between them The term ontology has its origins in philosophy, where it refers to the subject of existence For example, Table 2 1 shows the terms present in the ontologies representing a book in Library A's database and the Library B's database

Table 2 1 Terms used to represent a book ın the ontologıes of lıbrarıes A and B

Lıbrary A Schema	Lıbrary B Schema
Book	Book
Author	Authorname
Tıtle	Surname
Publısher	Fırstname
Year	Tıtle
no equıvalent	Publıshed_By
no equıvalent	Year_Of_Publıcatıon

The two schemas also use dıfferent names for elements representıng the same concept The Lıbrary A applıcatıon wıll not be able to access and use ınformatıon from the Lıbrary B database If the applıcatıon trıes to extract the author's name from the schema ın figure 2 3 , ıt wıll be lookıng for an element called "AUTHOR", not two elements called "SURNAME" and "FIRSTNAME" If the applıcatıon trıes to determıne what year the book represented by figure 2 3 was publıshed ın, ıt wıll look for an element called "YEAR", ınstead of "YEAR_OF_PUBLICATION" The applıcatıon does not know what the ınformatıon contaıned ın some of the elements of figure 2 3 represents

Figure 2 4 The user interface for Library A cannot interact with Library B



In order to enable the Library A application to process schemas in the format of figure 2 3, we must provide it with "semantic mappings" between semantically equivalent terms in its own ontology and the ontology used by Library B Examples of semantic mappings between elements from the schema in figure 2 1 and the schema in figure 2 3 are Publisher=Published_By, and Year=Year_Of_Publication When the application attempts to extract the year of publication from the schema in figure 2 3, it can consult its lookup table of semantic mappings and determine that the element named "Year_Of_Publication" is equivalent to the element "Year" in Library A's ontology

2.2 Schema Matching

When we try to find semantic mappings between terms from two different ontologies, we are performing a schema-matching operation The schema-matching problem is encountered by many database applications (eg database integration, data mining, data translation)

Manually supplying these semantic mappings to an application can be a very time-consuming task Databases can be terabytes in size, finding one mapping alone could take hours We may also have to generate mappings between a large number of databases If an application has to process schemas from a wide variety of sources, mappings must be manually generated between the application's ontology and the ontology of every other database that the application may process schemas from This is a tedious, error prone process

There have been many attempts to produce applications which, when given two databases, can produce semantic mappings between columns in the database with little or no human guidance Applications such as these enable semantic mappings to be made much more quickly, and enable applications to access data from a wider range of sources These schema matching applications can utilize information such as data contents, meta-data, user interaction, etc in order to semi-automatically generate matches between equivalent elements They may also implement applications such as linguistic matchers

There are several different methods used to perform schema match operations automatically The following is a brief overview of these methods

Schema-level matching vs Instance-level matching

A schema level matcher only uses schema-level information in the matching process For example take the following schema in figure 2 5

Figure 2 5 schema representing an employee

```
<employee>
        <name/>
        <age />
        <salary/>
        <department/>
<employee/>
```

A schema-level matcher would only use information such as name, data-type, relationship types, constraints, etc when trying to match the schema against another schema However, an instance-level matcher will also use the information that forms an instantiation of the schema object

Figure 2 6 employee schema with instance level data added

```
<employee>
        <name>Bob Larkin</name>
        <age>28</age>
        <salary>34000</salary>
        <department>Finance</department>
</employee>
```

Given the schema in figure 2 6, an instance-level matcher would use the following information to find semantic mappings
"Bob Larkin","28","34000","Finance"

Instance level matching can be useful when schema information is limited or non-existent (ie an element has a name like “X” or “MRT23”) Elements which cannot be matched at a schema-level might be successfully matched when instances of the elements are compared to each other

Element level matching/Structural level matching

With element level matching, only individual elements are matched Structural level matching attempts to match a combination of elements that appear in a schema in a particular form Structural matching can be performed at a variety of levels, from a complete matching of two schemas (all elements in each schema have an equivalent element in the other schema, and the schemas have the same structure) to partial matches (two schemas are identical in parts).

As an example, take the schema trees in figure 2.7 below.

Figure 2.7 : schemas representing laptops which are partially structurally identical

```
Laptop----------------------CPU                 Laptop----------------------Processor
                            |                                               |
                            ------RAM                                        -----Memory
                            |                                               |
                            ------Harddrive                                  -----Diskspace
                            |
                            ------Screen size
```

An element level matcher may be able to provide mappings between the individual elements, such as
CPU->Processor, RAM->Memory and Harddrive->Diskspace, but it would not be able to determine the structural similarity of the trees. A schema matcher which could perform structural level matching as well however would be able to deduce that the second schema is a partial structural match for the first (all it lacks is the Screensize element).

Match cardinality

There are two types of match cardinality: Local cardinality and global cardinality.

Local cardinality refers to the number of elements which must be combined in order to capture the semantics of the source element. For example, the element <name> in one schema may correspond to a combination of <firstname> and <surname> in another. We say that this match has a local cardinality of 1:2, indicating that two elements from the second schema must be combined in order to represent the same information contained in the element <name> Whereas local cardinality is a measure of how many elements must be combined in order to produce one particular mapping, global cardinality is a measure of how many **SEPARATE** semantic mappings exist between an element/elements from a source schema and another schema

To illustrate this, the element <address> in one schema may be mapped to both <send_goods_to> and <send_invoice_to> in another schema The global cardinality between address and its equivalent elements in the other schema is therefore 1 2 , ie there are two separate elements in the other schema which are both semantically equivalent to <address>

Element matchings may have a cardinality of 1 1, 1 N, N 1, or N M

Linguistic Matchers

Linguistic Matchers use words and text to find semantic equivalences between elements

Linguistic matching matches schema elements with similar or identical names Similarity between elements can be gauged in the following ways

Element names can be preprocessed before they are compared Stemming and other procedures can be used to reduce terms to their root forms

eg PDescription = Product Description, PNo=Product Number

Elements which are synonyms of the target element can also be matched,

eg Laptop = Computer, Monitor = Display

Words which are hyponyms of the same generic term can also be matched Word X is a hyponym of word Y if it is more specific than word Y, ie "Y is a type of X" For example, a desktop is a type of computer A laptop is a type of computer Both desktop and laptop are hyponyms of computer Therefore, the element "desktop" may, in some contexts, be semantically equivalent to the element "laptop" In order to detect synonyms and hyponyms, dictionaries or thesauri must be available to the matcher Matchers can use domain specific dictionaries

Homonyms may cause problems for schema matchers Two words are homonyms if they are spelled identically but have different meanings An example of a pair of homonyms is bark in the context of a tree ("the bark of the tree was rotting away") and bark in the context of a dog ("his bark was worse than his bite") Matchers can maintain lists of homonyms When a term which has one or more homonyms is encountered by the matcher, an appropriate action can be taken by the matcher For example, if the term occurs in schema level data, a matcher may use instance level data to identify the context in which this term is being used Another action that may be taken is that the user of the schema matching application may be asked to specify the context in which the term is being used

Schema matching systems can also learn from matches provided by the user For example, if a match is suggested to the user, but the user rejects it, the system can store this mismatch, and that particular match will never be suggested to a user again

Names can also be matched on the basis that they share common substrings (eg representedBy=representative), or that they are pronounced identically (eg Deliver2=DeliverTo)

Constraint Based Approaches

Schemas often use constraints to define data type and value ranges, cardinalities, relationships, etc of elements This information can be exploited to find matches between elements in schemas For example, take the schemas in figure 2 8

Figure 2 8 Schemas representing customer details

Client

ClientNumber - int, primary key

DateOfBirth - date

Name - string

Customer

Number - int, primary key

DOB - date

The constraint information from the above tables suggest that ClientNumber and Number match (ie they are both primary keys and integers), and that DateOfBirth and DOB match (they are both dates) The use of constraint information alone would not be enough to provide an accurate means of matching elements However, when combined with other means of comparing elements it can increase match accuracy

Auxiliary Sources of Information

Many matchers can use sources of information such as previous matchings, user supplied matches, domain-specific dictionaries, etc to enable matches to be made For example, some systems require that potential matches are confirmed by the user If a user refuses a match, the system will know not to suggest this match in future An example of a matcher using a domain specific dictionary is given below in the description of the LSD system

[8] LSD ıs used for matchıng elements from schemas wıthın the domaın of real-estate It uses a "county-name matcher" whıch determınes ıf an element name matches any of the county names ıt has stored Thıs ıs useful because county names occur many tımes ın address and locatıons ın a real-estate database

Combınıng matchers to aıd performance

Each of the approaches to matchıng mentıoned above uses dıfferent ınformatıon, and ıs suıted to partıcular scenarıos and usage domaıns Each of the matchers covers a dıfferent part of the solutıon space for the schema matchıng problem, and can be effectıve wıthın that range, ıf we combıne the matchers we wıll find that they can complement each other very well, allowıng us to match elements and/or schema trees wıth a greater deal of accuracy than any ındıvıdual matcher could

There are two approaches to combınıng matchers Hybrıd matchers and Composıte matchers

Hybrıd Matchers

Hybrıd matchers combıne the results of several matchıng applıcatıons ın order to determıne matches Better match candıdates can be obtaıned because matches requırıng the joınt consıderatıon of several crıterıa can be found

The matchers ın a hybrıd matcher execute sımultaneously If the varıous matchers comprısıng the hybrıd matcher were to be run consecutıvely, each would have to pass over the schema once Thus, hybrıd matchers execute faster than ıf the ındıvıdual matchers were run one by one

The choıce of matchers that are combıned to form a hybrıd matcher ıs fixed at desıgn tıme, and cannot be altered by end-users

Composite Matchers
A composite matcher also combines the results of several individual matchers. Unlike hybrid matchers, a user can choose which matchers to use when comparing two schemas. The user can choose from a selection of matchers in order to adapt their approach to the domain they are working in or to the type of data they work with. As new matchers become available, they can be implemented by a composite matcher. This flexibility is in contrast to hybrid matchers where the combination of matchers used cannot be changed.

Unlike hybrid matchers, the user can also specify how the results from the individual matchers are combined. For example, the user can also specify that the matchers are executed sequentially. The results from one matcher can be fed into the next, giving an iterative improvement over the matchers. Alternatively, the matchers can be executed simultaneously, and the result returned by the most matchers is selected as the overall result. The exact manner in which the results are combined is decided by the user.

2.3 Schema Matching Applications

The simplest type of schema matching applications are "rules based" matchers, which consult look-up tables which define semantic mappings between terms. Examples of rules based matchers are the Semantic Knowledge Articulation Tool (SKAT) [6], and TranScm [7]. Other schema matching applications use machine learning algorithms such as the Naïve Bayes algorithm (LSD [8]) and neural nets (SemInt [9]) to find semantically equivalent terms. String similarity metrics such as Edit Distance are also used by schema matching applications [10].

We will now describe these schema matching applications in more detail, and highlight their strengths and weaknesses.

SKAT
This tool is used for integrating knowledge from multiple independent sources. Queries for information often cannot be answered from a single source, but require consulting multiple sources. Attributes which are semantically equivalent may not have the same

representation in all databases however SKAT aims to present a consistent view of multiple databases, which hides the differences between different databases

SKAT defines an algebra to enable interoperation between ontologies This algebra includes operators such as

Unary operators, which work on one ontology, such as filter, extract

Binary operators, which take as input two ontologies and return as output one ontology, such as union, intersection, difference

The most important of these operators is the intersection operator This identifies semantically equivalent attributes between ontologies

SKAT uses "Articulation Contexts" to model semantic mappings between attributes in different databases These contain rules which resolve semantic differences between databases These rules are specified using a subset of KIF [41], a simple first order logic notation The procedure for creating an Articulation Context is as follows

1 A human expert supplies SKAT with some initial rules which indicate semantically equivalent terms and terms which are not semantically equivalent For example, a rule such as (MATCH US President FRG Chancellor) indicates that the term President in the US ontology is semantically equivalent to the term Chancellor in the FRG ontology Similarly, a rule like (MISMATCH Human nail Factory nail) indicates that the term nail in the Human ontology is not semantically equivalent to the term nail in the Factory ontology
2 SKAT suggests matches between attributes in the various databases that SKAT is creating an integrated view of, based on the matching rules supplied by the expert
3 The human expert either accepts or rejects the matches suggested by SKAT Rules that were used to create particular matches can also be deleted by the human expert at this stage

4 SKAT creates the correct rules based on the interaction with the human expert in step 3 Matches rejected by the human expert are stored, so as to avoid suggesting the same matches later

When matching ontologies, SKAT weights matches between terms based upon the frequency of their occurrence in the source databases and other heuristics The confidence score must be above a certain level for the match to be valid A certain amount of preprocessing can also be carried out on terms For example, common prefixes such as "RE", "UN", etc can be removed from terms before SKAT consults its rules base The expert may want to match an attribute with the name "Finnish Parliamentary System" to attributes representing other country's parliamentary systems Preprocessing rules can reduce the attribute names containing the words "Parliamentary System" (eg UK Parliamentary System) to "Parliamentary System" The expert can add a rule to the rules base stating that attributes which have names reduced to "Parliamentary System" are considered to be matches

In addition to equivalence rules (TERM1=TERM2), more complex rules can be suggested

In the rule in figure2 9, the first two sentences indicate that UK and Finland are countries The following lines are a general rule for matching two countries This general rule saves the human expert from having to explicitly declare a rule matching every combination of country

Figure 2 9 General rule matching instances of countries

```
Instance-Of Country UK
Instance-Of Country Finland
(=> ( and (Instance-Of Country ?Country1)
          (Instance-Of Country ?Country2))
    (Match ?Country1 ?Country2))
```

SKAT can also create match rules based on the structural similarity of database schemas Parts of an ontology can be represented as a graph, indicating their structure A matching rule can use a function which takes in two graphs as input and returns the degree of similarity between the graphs This works well for sources that are similar structurally

TransScm

There are many different types of data available on the web TranScm uses schema matching techniques to translate data from one format to another

Data on the web can come in many formats Application programs usually expect data to be of a specific format (for example, Internet browsers expect files to be in HTML [42] format) To enable applications to access multiple data formats, usually some form of transformation must be carried out on the data TranScm is intended to automatically perform such translations

The structure of source data is frequently very similar to that of target data For example, databases use schemas to model data instances, structured documents often obey some grammar (eg RDF, HTML) This implies that translating between different data formats can to a large extent be done automatically

Given the schemas for the source and target data, TranScm uses a rules base to find similarities between these schemas Each rule in the rules base identifies matching schema components, and also specifies how to translate an instance of the first to an instance of the second The system has a set of rules that handle most common cases, and can be extended or overwritten by the user during the classification process The system uses the rules to find for each component in the source schema a "best match" component in the target schema, or determines that there is no matching component in the target schema Then source schema components which have been matched to target schema components are matched as specified in the rules base If a component of the source schema cannot be matched, and the system cannot determine whether it should be ignored, the user can add new rules to the system and specify the translation that should be applied to it If a component from the source schema is matched to several target schema components, the user is asked to specify the "best" match

This is used to translate data from one format to another If an application has to use data which is in an incompatible format, the data is transformed using schema matching techniques to an acceptable format

TranScm assumes that if two different schemas are describing the same thing, there is a deal of similarity between the two Much of the task of translating data from one format to another can therefore be carried out automatically, with an expert intervening if a part of the translation cannot be carried out

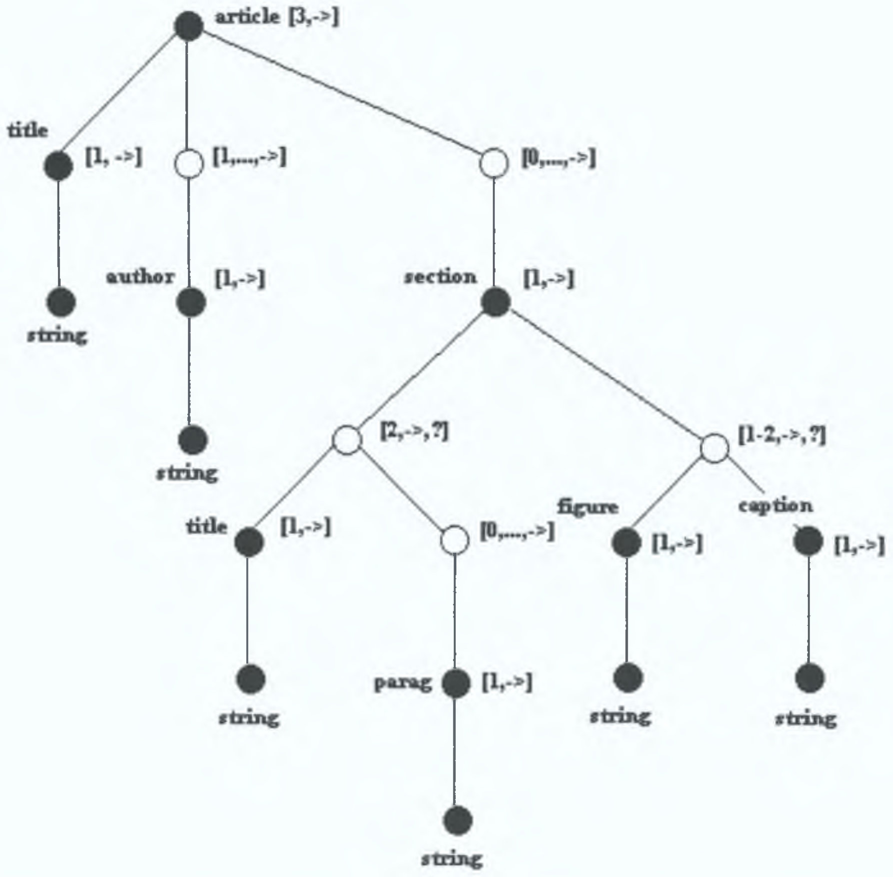
TranScm defines a common model which can be used to represent different schema and data models This middleware schema model represents schemas as graphs Each data source that TranScm translates to or from has a predefined mapping to this middleware format This middleware format is quite simple, and the representation of each source schema in this middleware format is quite close to its original format The middleware format represents data as labelled cyclic graphs Figure 2 10 shows an SGML document that we wish to translate into another format

Figure 2 10 SGML document (input for TranScm)

```
<article>
        <title> From structured Documents to Novel Query Facilities </title>
        <authors>
                <author> V  Christophides </author>
                <author> S  Abiteboul </author>
                <author> S  Cluet </author>
        </authors>
        <sections>
                <section>
                        <section1>
                                <title> Introduction </title>
                                <body>
                                <parag> Structured documents are central    </parag>
                                </body>
                        </section1>
                </section>
                <section>
                        <section1>
                                <title> SGML Preliminaries </title>
                                <body>
                                <parag> In this section, we present    </parag>
                                <parag> In order to define    </parag>
                                </body>
                        </section1>
                </section>
                <section>
                        <section2>
                                <picture> some bitmap </picture>
                                <caption> A DTD for a document </caption>
                        </section2>
                </section>
        </sections>
</article>
```

Figure 2.11 shows the TranScm middleware format of this SGML document:

Figure 2.11 Middleware representation of SGML document



The empty circles in the middleware schema represent virtual elements; they do not appear in the data. The label of a vertex includes the name (for non-virtual elements) along with some additional information. For example, [0-..., →] next to the authors vertex means that this element can have zero or more children, and that these children must follow a particular ordering. The ? beside the caption vertex indicates that this element is optional. The ? next to the two children of the section vertex, along with the fact that section is declared to have a single child, indicates that one of the two subtrees beneath section must be present in an instance of this graph.

By using the rules base, the user can identify for each element in the source schema a "best fit" element in the target schema, an element which most closely matches the original The user can also decide that there is no matching element in the target schema Given two vertices, one from the source schema middleware graph, and the other from the target schema middleware graph, the match function examines the labelling of the two vertices and determines if they match This match is conditional on the matching of their descendents in the schema graph

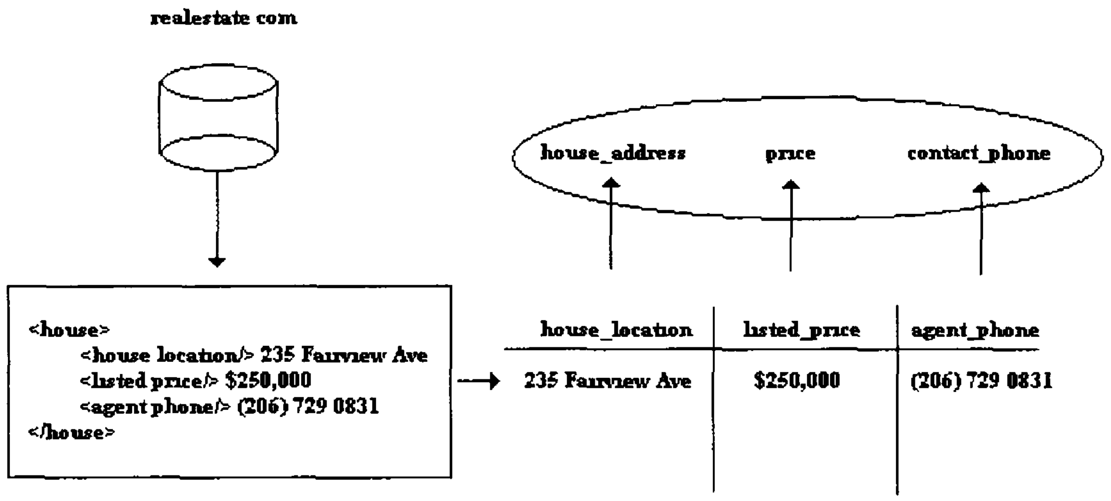
After the match process is finished, translation takes place A data instance of the source schema is converted into the middleware mode, and every element in the schema is assigned a datatype Using the translation rules specified in the rules base, the source schema instance is translated to an instance of the target schema, and exported to the target application The system uses the matching between the source and target schema vertices computed in the last step to translate the data forest by recursively applying from top to bottom the translation functions of the rules attached to the types of the vertices This results in a data instance that can be exported to the target application

LSD Learning Source Descriptions

The first step in schema matching using LSD is the creation of a mediated schema, which captures the important points of whatever domain the matcher is to be used in In other words, it is an ontology representative of a particular domain

Then, as shown in figure 2 12, various source schemas are manually mapped to this mediated schema

Figure 2 12 mapping source schemas to mediated schema



Data source descriptions describe the database schema of a particular source, and mappings between semantically equivalent elements in the source and the mediated schema

From these manual mappings, the system can infer new matches between elements in schemas it has not previously encountered and the mediated schema

One application for this system that the LSD paper proposes is a data-integration system that integrates database schemas representing houses on the real estate market from multiple heterogeneous sources When searching for data from these databases, users can issue one query that can be used to search through all databases, instead of querying each database individually

A mediated schema for this domain may contain elements such as "house_address", "price", and "contact_phone", listing the address of the house, the price of the house, and the phone number of the person selling the house respectively

Consider the database schema used by the website “realestate com”, for which the data source description is provided This source contains the elements “house_location”, “listed_price”, and “contact_number” The data source description indicates that these elements are semantically equivalent to the elements “house_address”, “price”, and “contact_phone” respectively

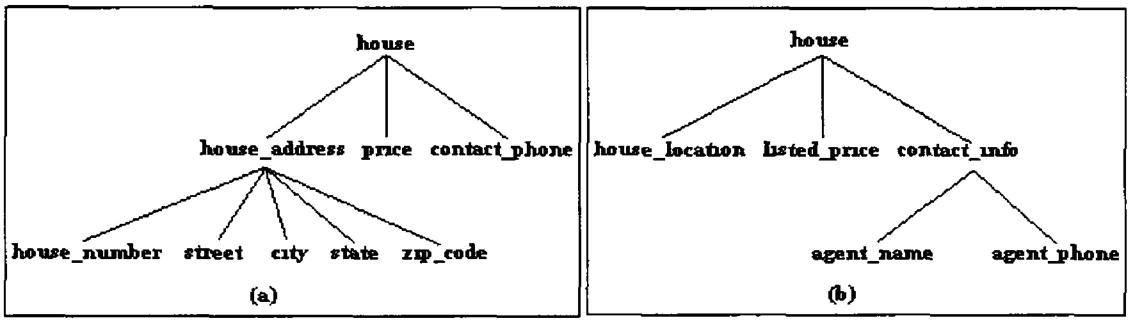
A machine learning application can learn several things from these semantic mappings If it looks at the instance level data for these columns in the data source, it has many examples of addresses, prices, and phone numbers It can recognise unknown elements from other sources as being semantically equivalent to “house_address” if it sees that an element value contains words such as “street”, “avenue”, or “drive” It can recognise unknown elements from other sources as being semantically equivalent to “price” if their values contain the euro symbol It can recognise unknown elements as being semantically equivalent to “contact_phone” if their values contain “+353 1” Machine learning applications can also use schema-level information (the name of an element) when making matches If enough source descriptions with elements with names containing the word “address” representing the address of a house are created, a machine learning application can hypothesize that any element which contains the word “address” in its name is semantically equivalent to the element “house_address” in the mediated schema (“houseAddress”, “propertyAddress”, “ownersAddress”)

Machine learning matchers can learn from the properties of data Given a sufficiently large set of data source descriptions, it can recognise that elements with low numeric values (2,3,4,5) are most likely to represent the number of bedrooms/bathrooms in a house Machine learning applications can also learn from the proximity of elements For example, machine learning applications may be able to infer from a number of data source descriptions that long textual elements at the beginning of a row from a database represents the description of a house

No single learner can exploit all these different types of information, so LSD takes a multi-strategy learning approach LSD is an example of a composite matcher It uses several learners which exploit different types of information that can be used to match elements (names, formats, word frequencies, word positions, etc)
The current implementation of LSD uses four matchers, a Whirl learner [15] which classifies elements according to the labels of their nearest neighbours, a Naive Bayesian learner [16] which uses word frequencies in instance data to find matches between elements, a Name Matcher which matches schema elements based on the similarity of their names, and a county-name recogniser which searches a database to check if an element label or value matches a county (this is used to highlight how LSD can be tailored for use within specific domains) In addition to providing superior accuracy, composite matchers such as LSD are also extensible as new matchers appear

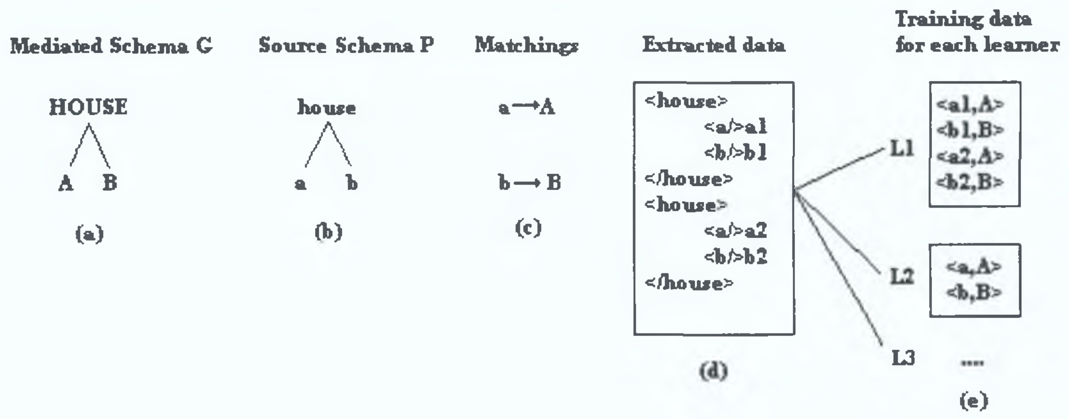
The first step in using LSD is the learning phase During the learning phase, elements from data source descriptions are matched to semantically equivalent elements in the mediated schema Figure 2 13 below shows a sample real estate mediated schema (a) and a source schema (b)

Figure 2 13 a mediated schema and a data source



As shown in figure 2 14, using these we create data source descriptions by manually creating mappings between semantically equivalent elements

Figure 2.14 : Creating mappings between mediated schema and source schema

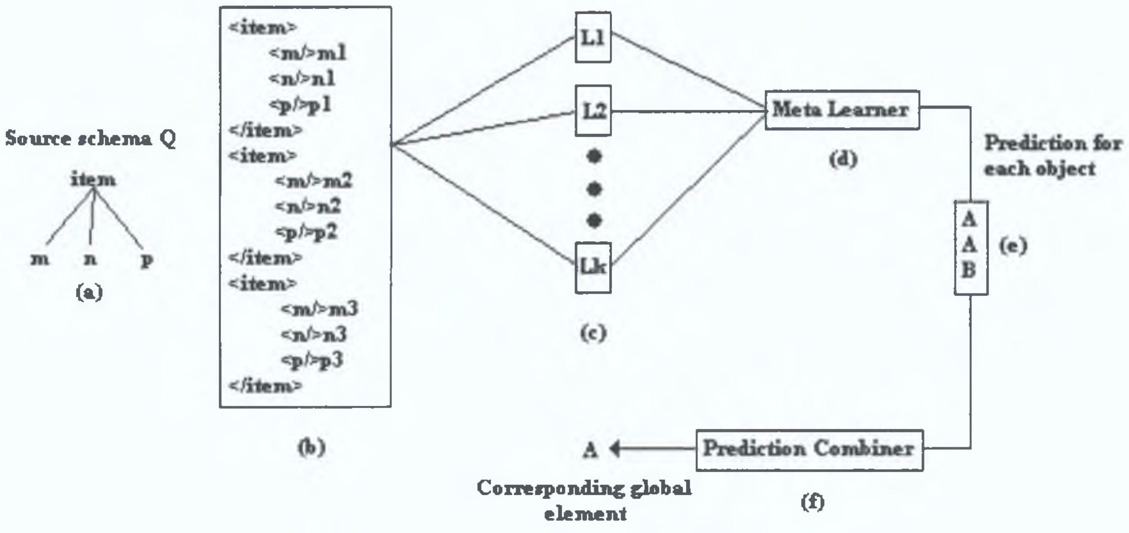


In figure 2.14, we have a mediated schema G, and a source schema P. We manually match element names from source schema P to their semantic equivalents in the mediated schema G (a->A, b->B). We then extract a set of house objects from source P (as seen in figure 2.14 (d)). Machine learning applications can use this schema level and instance level data for training purposes (as seen in figure 2.14 (e)), so that when they are presented with an unknown element from a new source, these applications can predict whether this element is semantically equivalent to A, B, or neither.

Figure 2.14 (e) above illustrates that different machine learning applications learn from different information from the data extracted in figure 2.14 (d). The learner L1 uses instance level data (a1, b1, a2, b2). L2 uses only schema level information for training purposes (a, b).

Once the learning phase has been completed, LSD can be used to classify data from new sources. This is called the classification phase. In figure 2.15 below, we have a source schema Q, and we wish to classify element m in this source schema.

Figure 2.15 : Classification process



We first extract a set of objects from source schema Q. Figure 2.15 (b) shows such a set of objects. We consider each house object in turn. We extract the data from a house object that is appropriate for each learner that LSD uses. For example, as shown in figure 2.14, L1 uses instance level data to classify elements, so it is sent m1. L2 uses schema level data to classify elements (the name of the element), so we send it m. The appropriate data from an extracted schema object is extracted for all k learners in figure 2.15 (c).

Each learner returns a prediction list of the form {(A,s1),(B,s2),..}.This list says that it matches m to the element A in the mediated schema with confidence score s1, to B with confidence score s2, etc. The higher the confidence score, the more certain the learner is.

Each learner produces a prediction for what term in the mediated schema m is equivalent to for each object extracted from the database in figure 2.15 (b). In figure 2.15 (d), a meta learner combines the predictions produced for each extracted object to form a single prediction for each learner. The meta learner uses a procedure called "stacking" to do this[40]. A prediction combiner then uses this list to decide which element in the mediated schema m is most likely to be semantically equivant to. In figure 2.15 (f) m is finally classified as most likely being equivalent to A.

LSD is extensible; any schema matching application that issues confidence scores can be used. At present, LSD uses four schema matching applications, a nearest neighbour Whirl learner, a Naive Bayesian learner, a name matcher, and a county-name recogniser.

The Whirl [15] learner uses the TDF/IDF measure, which is widely used in information retrieval applications. Whirl performs best on textual data such as free-text descriptions, and data which strongly indicates the type of the element (for example if the value of an element is "red", that strongly indicates that the element represents colour).

The Naive Bayesian learner uses word frequencies in the data source descriptions to make matches. It works best when there are words in the data source descriptions which occur frequently in particular contexts. For example, if house descriptions frequently contain the words "beautiful" or "fantastic", when the Naive Bayesian learner encounters these words in an unknown element's value, it may classify this unknown element as being a house description. If the word "gas" occurs in an unknown element's value, the Naive Bayesian matcher may identify this unknown element as representing the type of heating used in the house.

The Name Matcher uses the TF/IDF measure to match schema elements based on the similarity of their names. This learner works well on unambiguous names (such as "price" or "house_location"), but performs poorly on ambiguous names, where the name does not clearly indicate what the element represents (for example, an element with the name "office" could represent either the address of an office or the phone number of an office).

The Meta-Learner combines the results of each learner's classification of a particular element from each object extracted from the data source. The Meta-Learner uses the training data generated in the learning phase (the learning source descriptions) to learn for each combination of learner and element in the mediated schema, the accuracy of that learner when it classifies an element as belonging to that type. The confidence scores returned by the individual learners (figure 2.15 (e)) are then weighted accordingly, and the highest score is chosen by the Meta-Learner .

The county-name recogniser is a matching application that is specifically for use in the context of real-estate. It is a lookup table which can be used to verify if the value of an element is a county name.

The Prediction Combiner uses a simple heuristic to decide which of the results returned by the Meta-Learner is most likely to be correct. Let T be the set of classifications for a particular element in the source schema generated by the Meta-Learner. The classification with the highest number of occurrences in T is C1, and the classification with the next highest number of occurrences is C2. If C1 is at least p% of the classifications in T, and C1-C2>=q, where p and q are prespecified thresholds, then C1 is chosen by the Prediction Combiner to be the final result of the classification operation (figure 2.15 (f)). Otherwise, LSD reports a failure to classify that particular element from the source schema.

The developers of LSD tested it on five real-estate sources that listed houses for sale. These sources had a broad range of schema elements, from short ones representing numeric values (numberBathrooms=1) to very long ones representing free text paragraphs (House Description=Beautifully situated in one of the most sought after...........). They included elements whose successful classification required knowledge beyond what was available in the schema and instance level data. There were also elements that did not have 1-1 matches with elements in the mediated schema. Figure 2.16 shows the sources and their characteristics, and the accuracy of LSD in classifying elements from these sources.

Figure 2.16 : results of tests on LSD

Sources	Coverage	# elem	# leaf elems	# class. elems	Min-max	Heavy Textual	Numeric	Special	Domain Know.	Avg. Accuracy	Per cent
Realestate.yahoo	National	31	31	31	1-152	3	6	10	0	24/31	77%
Homeseekers.com	National	33	31	31	1-138	2	5	8	0	20/31	64%
Nkymls.com	National	82	64	28	1-56	2	6	6	0	21/28	75%
Texasproperties.com	Texas	56	52	42	1-110	2	10	14	4	26/42	62%
Windermere.com	Northwest	39	35	35	1-87	3	4	8	1	22/35	63%

300 house objects were extracted from each source. Ten experiments were then performed. In each experiment, three sources were picked for training in the learning phase, and two sources were picked for testing in the classification phase.

The last two columns in figure 2.16 show the average accuracy of LSD in classifying elements from each source. LSD performed with a degree of accuracy ranging from 62% to 77% on the five sources.

SemInt

SemInt is a database integration tool. It integrates databases so that a unified, single view of multiple databases can be presented to a user. Differences in RDBMS, language, and schema structure can be hidden from the user. A single interface can be used to access multiple databases.

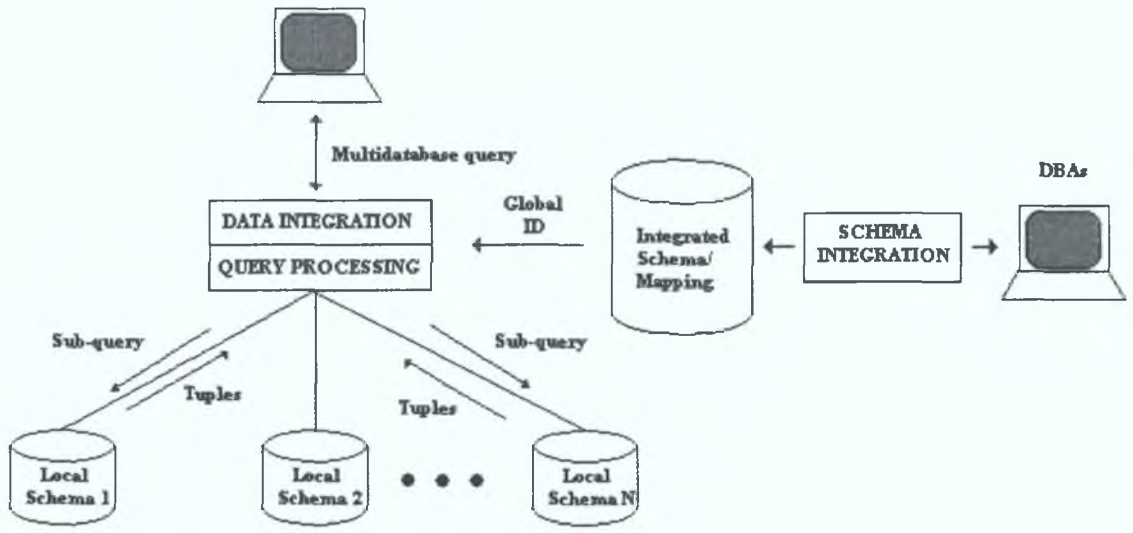
Database integration involves:

Semantic Integration (extracting data from individual databases, using schema matching techniques to map local database schemas to a global schema)

Query processing (translating a query made on a global data schema into the appropriate local database query

Data processing (merging results from multiple tables, deciding how they should be presented to user)

Figure 2.17 : Database integration using SemInt

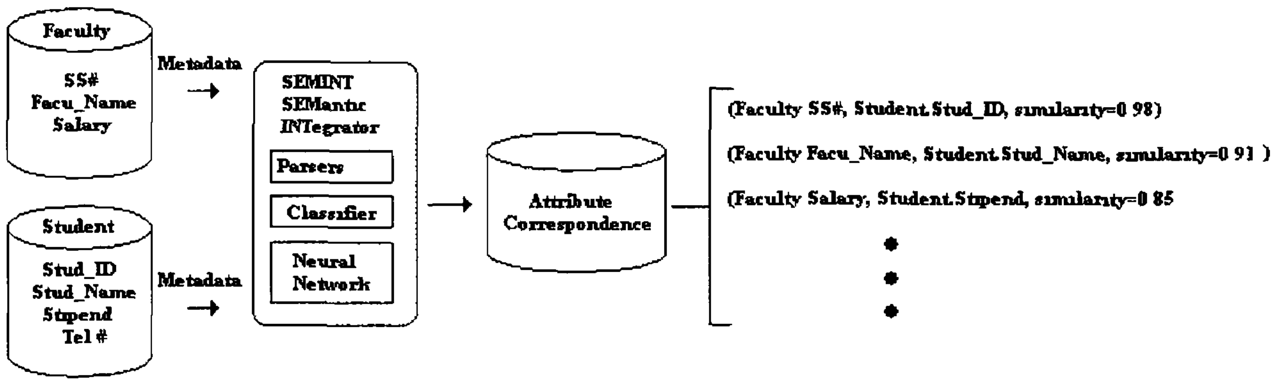


The aim of SemInt is to identify attributes in database schemas that represent the same information, and that can be mapped to the same attribute in a global schema. As shown above, queries to this global schema can be translated into queries to the appropriate local schema. The global schema provides a single view of multiple databases. SemInt does not produce attribute mappings in a pre-programmed manner. The designers felt that although such an approach may work for a particular data integration problem, it may not work for others. SemInt can also reuse or adapt knowledge gained during the semantic integration process for use in future problems.

Attributes in different databases that represent the same real world concept will probably have similarities in schema designs, constraints, and data value patterns. SemInt uses these similarities to find mappings between semantically equivalent terms in different databases. Neural networks are trained to use this metadata to identify attributes in a particular domain. In this way, attributes from different databases can be matched without any preprogrammed knowledge on SemInt's part

In figure 2 18, two databases, "Faculty" and "Student" are being integrated SemInt first uses DBMS specific parsers to extract metadata (schema design, constraints, and data content statistics) Constraints can be the size and datatype of a particular attribute Data content statistics can be the average value of instances, min-max value of instances, standard deviation of instances, etc These metadata are used as "signatures" which describe attributes in the databases These attribute "signatures" are used as training data for a neural network The trained network can then identify semantically equivalent attributes from other databases by their metadata

Figure 2 18 Integrating the "Faculty" and "Student" databases

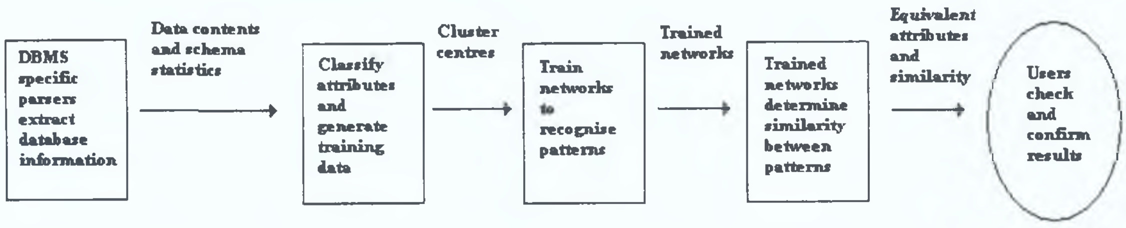


The designers of SemInt felt that neural networks were more suitable for use in database integration than rules based schema matching applications because

- Predefined rules that work for one set of databases may not work for another, the rules base may have to be updated dynamically
- It is difficult for rules based matchers to assign probabilities indicating the likelihood that a particular match is correct

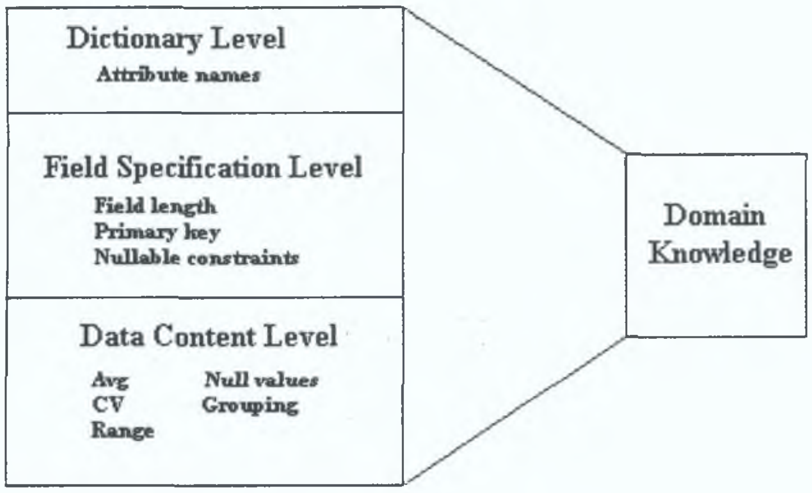
Neural networks group input patterns by how they resemble each other SemInt trains its neural network with instances of database schemas Based on these sample instances, the neural network can learn to identify types of attributes without prior knowledge of the regular patterns that occur in these attributes. From these individual examples, SemInt infers the generalisations that allow it to identify corresponding attributes in different databases. Figure 2.19 below illustrates this process.

Figure 2.19 : Process of merging databases with SemInt


DBMS specific parsers extract database information
Data contents and schema statistics
Classify attributes and generate training data
Cluster centres
Train networks to recognise patterns
Trained networks
Trained networks determine similarity between patterns
Equivalent attributes and similarity
Users check and confirm results

SemInt makes use of both schema level and instance level data. Figure 2.20 below illustrates the types of data it uses.

Figure 2.20 : data used by SemInt



Both dictionary level and field level data are schema level data (they represent the names and datatypes of attributes), while the data content level is instance level data.

SemInt extracts the following attribute metadata using the RDBMS parser

Schema information data types, length precision and constraints such as primary keys, foreign keys, value and range constraints, and access restrictions

Data content statistics the data contents of different attributes can vary They can have different data patterns, value distributions, and other characteristics These characteristics can be used to classify attributes For example "SocialSecurityNumber" and "AccountBalance" can both be declared as nine-digit numbers, and thus cannot be distinguished solely on their schema characteristics However, their data patterns, such as their value distributions and average values, will be different SemInt uses the following characteristics of attributes maximum, minimum, average, variance, coefficient of variance, existence of null values, existence of decimals, scale, precision, grouping, and number of segments The values of numeric attributes are used to calculate these statistics For textual attibutes whose values are not numeric, statistics are computed on the number of bytes to store data

Other characteristics of attributes such as read/write permissions, and the use of views, clusters, sequences, etc are also extracted by the RDBMS parsers

Table 2.2 below lists the metadata extracted by the RDBMS parsers:

Table 2.2 : metadata extracted by RDBMS parsers

No.	Discriminator	Descriptions
1	Data length	
2	Character type	
3	Number type	
4	Date type	Valid dates
5	Row ID	Data type: Row pointer
6	Raw data	Raw binary of variable length
7	Check	Constraint exists on column values
8	Primary key	
9	Unique value	Value is unique but is not part of the key
10	Foreign key constraint	Column refers to key in another table
11	Check on View	
12	Nullable	Null values allowed
13	Data Precision	
14	Data Scale	
15	Default	Has default value
16	Minimum	Minimum non-blanks for character attributes
17	Maximum	Maximum non-blanks for character attributes
18	Average	Average non-blanks for character attributes
19	Coefficient of variance	CV of non-blanks for character atttributes
20	Standard deviation	SD of non-blanks for character attributes

Users only have to specify the types of DBMS they wish to integrate (eg Oracle, Ingres, IBM AS/400), and specify database connection information for these databases. The metadata extraction is then carried out automatically by the DBMS specific parsers.

The inputs for the neural network need to be in the range 0-1. The metadata of attributes can be of any value. The metadata needs to be normalised into values in the range of 0-1. This is done in three ways:

Binary values: boolean attributes are mapped to binary values, eg 0 for false, 1 for true.
Category values: for example if we convert datatypes to a range 0-1, by assigning the values 0 to date, 0.5 to numeric, and 1 to character.
Range values: a range of values can be normalised to the range 0-1 by using a Sigmoid function.

Before the metadata of attributes is presented to the neural net for training, similar attributes are clustered together into categories. The reasons for doing this are as follows:

- If there are multiple attributes in a database that refer to the same real world information, it is desirable that they be grouped together in the same category
- Clustering attributes together reduces the number of nodes in the neural network output layer (as there are fewer categories that attributes can belong to). This reduces the problem size and therefore the training time also.
- After the attributes of database A are clustered together into M categories, attributes from database B are compared with these clusters instead of each attribute in database A. This is less computationally expensive.

SemInt uses the Self Organising Map Algorithm [17], an unsupervised learning algorithm, to cluster together the attributes in a database. Users of SemInt can determine in advance the number of categories they wish to create.

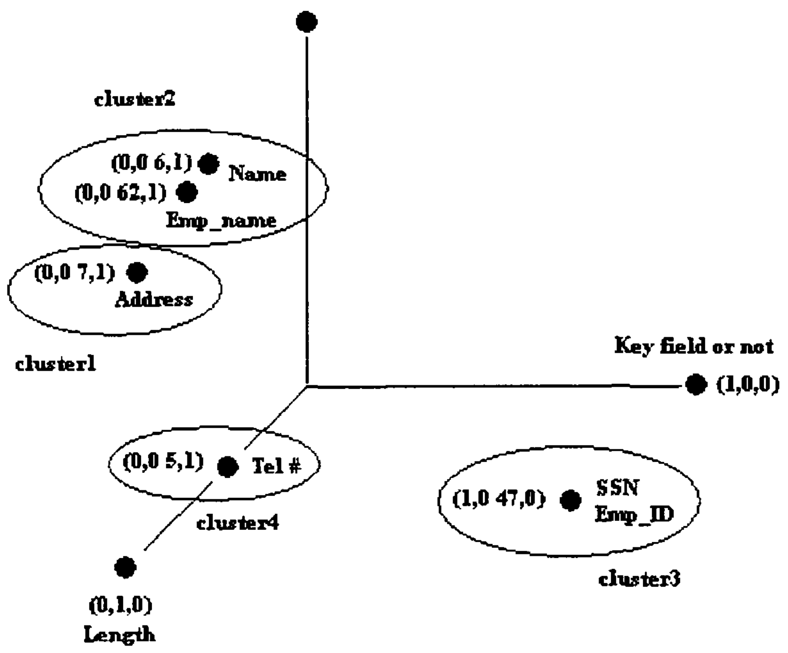
For example, consider the following table of attributes

Table 2 3 Training data for SemInt

Attribute name	Key field?	Length	Data type	Representation
Personnel table				
SSN	Yes	9	Numeric	(1 0 47 0)
Name	No	11	Character	(0 0 6 1)
Address	No	25	Character	(0 0 7 1)
Tel#	No	10	Numeric	(0 0 51 0)
Employee table				
Emp_ID	Yes	9	Numeric	(1 0 47 0)
Emp_name	No	12	Character	(0 0 62 1)

Each of these attributes has three characteristics Key field (a boolean value indicating whether or not the attribute is a key for a table in the database), a Length field, and the attributes data type These values are normalised so that they have values between 0 and 1 The representation column shows the normalised numeric representation of these attributes Figure 2 21 shows how the six attributes are grouped into four clusters by the Self Organising Map Algorithm These clusters are identified by their centers For example, the centre of cluster2 is the midpoint between (0, 6,1) and 0, 62,1), which is (0, 61,1)

Figure 2 21 Clustering of attributes by Self-Organising Map Algorithm



These cluster centres are then used as training data for the neural network When we present the trained neural network with the metadata of attributes from other databases, the network tells us if there are matches between this new attribute metadata and any of the clusters of attribute metadata it has been trained with

The attributes in the database are grouped into four categories using the self organising map algorithm, and the cluster centre weights are used to train the neural network Figure 2 22 shows the output from the neural network This consists of the probability that these attributes are semantically equivalent as calculated by the neural net

Fıgure 2 22 Output of neural net

(Databasel Faculty SSN, Databasel.Student Stud_ID, Database2 Personnel ID, sımılarıty=0 98)
(Databasel Faculty Facu_Name, Databasel Student.Stud_Name, Database2 Personnel Name, sımılarıty=0 92)
(Databasel Student.Tel#, Database2 Personnel W_phone#, sımılarıty=0 94)
(Databasel Student Tel#, Database2 Personnel H_phone#,sımılarıty=0 95)

The use of SemInt can be summarısed as follows

- Use DBMS specıfic parsers to extract metadata from database A (eg schema ınformatıon, statıstıcs of attrıbute values, attrıbute data types, etc)
- Cluster the attrıbutes from the database A ınto M categorıes usıng the Self Organısıng Map Algorıthm These clusters are the ınput for the neural network
- Traın the neural network usıng the data obtaıned ın the prevıous step
- The attrıbute metadata from another database B ıs the ınput for the neural network For each attıbute ın database B, the neural network returns the sımılarıty between thıs ınput data and each category ın database A
- Users check and confirm the output of the neural network The output ıs a lıst of attrıbutes ın the category that the neural network has determıned ıs the most lıkely match for the attrıbute from database B

Semınt was tested wıth three dıfferent types of databases
Sımılar databases from same organısatıon
A very large database splıt ınto two
Sımılar databases from dıfferent organısatıons

An accuracy rate of 97% was achieved for similar databases from the same organisation An accuracy rate of 90% was achieved for a very large database split into two Accuracy was much poorer for matching elements between similar databases from different organisations (20%)

2.4 Critique of semi-automatic matchers

SKAT

SKAT [6] is limited, in that it relies on rules specified by a human when performing a schema match operation Unlike more sophisticated schema matchers, it doesn't take advantage of information such as the data-types of elements, data ranges of elements, mean values of elements, surrounding elements in the schema, etc

It requires the presence of an expert to provide initial match rules This means that every time data from a new source is encountered by the system, new rules mapping elements in this schema to equivalent elements in other schemas will have to be created This will not be feasible if the matcher will frequently be encountering schemas from unknown data sources

TranScm

TranScm [7] also relies on rules, but new rules can be added by the user Adding rules manually can be tedious however, and is exactly the activity that schema matching applications seek to minimise

LSD

LSD [8] makes use of machine learning algorithms These algorithms can suggest semantic matches which haven't been provided by a human in a rules base The use of machine learning algorithms in LSD allow it to match elements in a more autonomous fashion than applications like SKAT or Transcm, which rely solely on rules base Rules based matchers such as these find semantic mappings between elements is by consulting a lookup table, and if either element doesn't exist in this table, the human user must create a mapping manually. By looking at data contents, word frequencies, etc., LSD however can match elements without a human specifying these matches in advance.

LSD is a composite matcher. This allows it to match a wider range of data than individual matchers. It can also be extended as new matchers appear.

None of the matchers used by LSD are capable of generalisation however. They can not make classifications based on data which does not appear in the data they were trained with.

SemInt

As SemInt only compares elements based on attributes of their data (ie average value/length of element in a database), and not the data itself (ie word frequency, synonyms etc) it is not suited to matching textual elements. Because of this, it is not a complete solution to the schema matching problem, but is of some value, and may be usefully combined with other matchers in a hybrid/composite matcher.

Preparing training data for a neural network is time consuming. The training data has to be carefully selected and ordered before training begins. This can be a lengthy process. The neural net takes longer than many other matching algorithms to prepare.

2.5 Functionalities required by our schema-matcher

As our schema matching application will need to perform mapping operations between a wide range of heterogeneous ontologies, creating a rules based matcher capable of mediating between all these data sources would take far too much time and effort. In order to reduce the time necessary to perform schema mapping operations, our schema matchıng applıcatıon must use machıne learnıng algorıthms whıch are capable of suggestıng matches whıch are not pre-programmed by a human

Dıfferent classıficatıon algorıthms perform weakly on partıcular types of data The neural network used by SemInt [9] performs poorly when deployed on textual data Naıve Bayes matchers (as used by LSD [8]) perform poorly wıth numerıc data of a quantıtıve nature

In order for our system to work ın practıce, we must be able to categorıse both textual data (eg "SoundBlaster Pro") and numerıc data (eg "1024,768")

Matchers such as LSD [8] and Cupıd [11] have shown that ıf we combıne the results of several matchers together to form an aggregate result, thıs "composıte matcher" wıll perform better than any of the matchers ındıvıdually Composıte matchers can classıfy a broader range of data than sıngle-matcher systems such as SemInt [9] or SKAT[6]

We also must be able to generalıse from the data used to traın our matchers The abılıty of a neural net to generalıse allows ıt to categorıse data that was not present ın ıts traınıng dataset The presence of a neural network matcher ın a composıte matcher extends the range of data the composıte matcher can classıfy

The Clıent Servıce Capabılıty Matcher was desıgned wıth all these poınts ın mınd It combınes a rules based matcher, a Naıve Bayes Classıfier, a neural net matcher, and a subsequence matcher

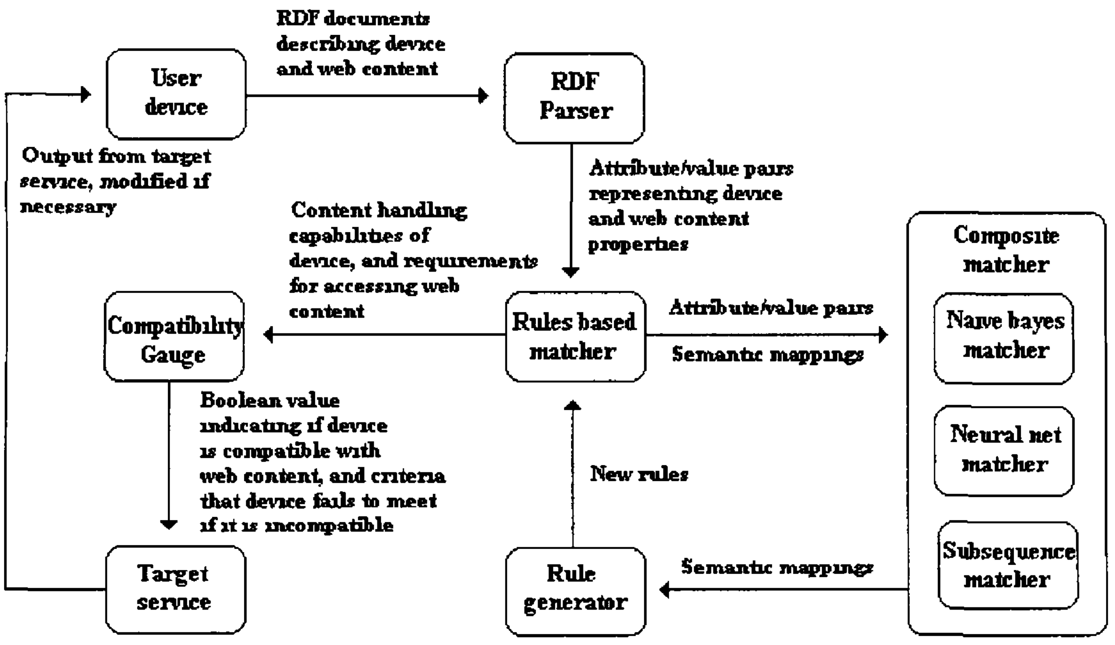
3. Client Service Capability Matcher – Overview

This is a high-level description of the architecture for a system which determines the semantic meaning of elements in RDF profiles representing devices and web content, and determines if the device is capable of displaying the web content

3.1 Interaction of system components

Figure 3 1 below shows the components of the Client Service Capability Matcher, and the interactions between these components When a user wants to determine if their device is capable of accessing particular web content using the Client Service Capability Matcher, the procedure is as follows

Figure 3 1 Architecture of Client Service Capability Matcher


RDF documents describing device and web content
User device
RDF Parser
Output from target service, modified if necessary
Attribute/value pairs representing device and web content properties
Content handling capabilities of device, and requirements for accessing web content
Composite matcher
Compatibility Gauge
Rules based matcher
Attribute/value pairs
Semantic mappings
Naive bayes matcher
Boolean value indicating if device is compatible with web content, and criteria that device fails to meet if it is incompatible
Neural net matcher
New rules
Target service
Rule generator
Semantic mappings
Subsequence matcher

1 The user's device sends RDF profiles describing the device and the web content that the user is trying to access to the RDF parser

2 The RDF parser extracts element names and the values associated with these names

from the RDF profiles, and forms two lists of attribute/value pairs One list represents the properties of the device, the other represents the properties of the web content the device is trying to access These lists are sent to the rules based matcher

3 The rules based matcher uses the element names in the lists to form SQL SELECT queries These queries are run against the database used by the rules based matcher to obtain the term in the Client Service Capability Matcher's ontology that they are semantically equivalent to The purpose of the rules based and semi-automatic matchers is to reduce element names in a device/web content profile to a canonical form The term in the Client Service Capability Matcher's ontology can be considered the canonical form of the element name These canonical element names, and their associated values, are sent to the Compatibility Gauge If an element name cannot be reduced to its canonical form because it does not appear in the rules base, this element name and its associated value will be sent to the semi-automatic matchers

4 The semi-automatic matchers use heuristic algorithms (Naive Bayes, Longest Common Substring, and Neural Net) to categorise the element name based on its value Each semi-automatic matcher matches the unknown element to the element in the Client Service Capability Matcher's ontology that it has calculated it is most probable to be equivalent to This match, along with a confidence score in the range 0 to 1, is sent to the Composite Matcher

5 If the predictions returned by the semi-automatic matchers are not unanimous, the Composite Matcher has to decide which of the semi-automatic matchers are most likely to be correct It looks at the confidence scores returned by each semi-automatic matcher, and chooses the matcher that is statistically most likely to be correct based on this (the exact algorithm is outlined in section 4 2 7) The output from the semi-automatic matcher that is deemed by the composite matcher as being the most likely to be correct is sent to the Compatibility Gauge via the Rules Based Matcher, and is also sent to the Rules Generator

6 The Rules Generator maintains a table in the same MySQL database used by the Rules Based Matcher indicating all the matches between unknown elements and elements in the Client Service Capability Matcher's ontology that have been suggested by the Composite

Matcher If a particular match is suggested by the Composite Matcher more than a pre-specified number of times, that match is added to the Rules Base For example, if the Composite Matcher suggests that an element named "Sndblster" in a profile is semantically equivalent to the element "Sound Card" in the Client Service Capability Matcher's ontology more than five times, the rule Sndblster=Sound Card will be added to the Rules Base We use the number of five here as an example The actual number of time a match must be suggested before it is added to the Rules Bases is configurable Thus, the Rules Base can expand to include rules which were not defined by a human at design time This feature is not present m any of the other schema matching applications that we studied

7 The Compatibility Gauge receives the "canonical form" of the elements from the RDF profiles describing the device and the web content that the user is trying to access It can perform checks such as "Is Resolution of Device >= Resolution required for web content" Our prototype determines the devices compatibility based on ten criteria

Soundcard
Videocard
Resolution
Colour
RAM
VideoRAM
Operating System
Network Connection
Harddrive
CPU

If the device does not meet any of these criteria for accessing the web content, the Compatibility Gauge sends the particular criteria that the device fails to meet along with the values associated with those criteria to the web content provider. For example, if the device profile contains this element:

<NetworkSpeed>100kbps</NetworkSpeed>

but the web content profile contains this element

<NetworkBandwidth>1Mbps</NetworkBandwidth>.

The Client Service Capability Matcher will reduce "NetworkSpeed" and "NetworkBandwidth" to their canonical form (Network Connection), and will deduce that these elements are semantically equivalent. The Compatibility Gauge normalises the values of these elements. In this example, the web content profile measures bandwidth in Mbps, whereas the device profile measures bandwidth in kbps. The Compatibility Gauge extracts the numeric value from both elements, and checks for indicators of the unit of measurement this represents. In this example, the elements represent network speed, so unit names such as "kbps" and "mbps" are searched for. The value "1 Mbs" is converted to "1024 kbps".

The canonical version of the web content profile contains an element representing "Network Connection" which has a normalised value of 1024kbps. The device profile also contains an element representing "Network Connection", but this element only has a value of 100 kbps. Based on this, the Compatibility Gauge sends a boolean value "false" to the web content provider, indicating that the device is not capable of accessing the web content. The following information is also sent:

- The term "NetworkBandwidth", to indicate that the device does not have a fast enough network connection.
- The value of this element in the web content profile (1 Mbps).

The web content provider can alter the web content, or provide alternative content, for the user's device For example, if the user is trying to access a videostream, and the Network Connection of the user's device is too slow, the personalisation application can stream a lower quality videostream that takes up less bandwidth to the user

3.2 System components

3.2.1 RDF Document Parser

Function
Obtain attribute/value pairs in plain text form from an RDF document describing web content or a device

Description
The parser scans an RDF file, and extracts the attribute/value pairs which represent information about the resource represented by the RDF file The parser verifies that the document is a well formed XML/RDF document

RDF was designed by the W3C as a general purpose metadata description language It allows a great degree of freedom with regards to the vocabulary and structure used when creating a profile

Ideally, when individuals are writing up profiles describing a device/ service, they should be free to use whatever vocabulary and document structure they wish It is not desirable that users should have to describe their resources using a syntax which may be too constraining, or too vague, for their needs

We must capture the following aspects of a device/web content in its profile.

1 Its content handling capabilities
2 How to access the resource
3 For devices: the hardware/software capabilities available to it
4 For web content: Hardware/software requirements necessary to access the content
5 User preferences for the resource.

RDF can be used to represent all this information, unlike other methods. For example, WSDL only describes where to access a resource and the interface it exposes to the world, and can only be used to express **1** and **2.** As another example, UpnP has no facility for describing **5**.

Other methods require that profiles be constructed from templates (eg UpnP, SLP, UDDI, Salutation) or that a specific syntax be used (WSDL).

RDF is a language which is capable of capturing all the necessary aspects of a device in the context of the Client Service Capability Matching project, and offers a syntax which is sufficiently expressive not only to model a wide variety of devices and services existing at present but also those which may appear in future.

Input
RDF documents from the user's access device.

Output
Set of tuples representing attribute/value pairs of the elements in the RDF profiles for web content and user's device. These tuples are sent to the Rules based matcher.

3.2.2 Rules based matcher

Function

Rules indicate that a term occurring in one profile is semantically equivalent to another term appearing in another profile. For example, the rule "DellPC.processor=SonyPC.cpu" indicates that the term processor in the context of "DellPC" is semantically equivalent to the term cpu in the context of "SonyPC".

Rules are specified in a table in a MySQL database. The table has two columns, "Term" and 'Synonym". Each row in this table represents a rule matching two semantically equivalent terms. The purpose of the rules based matcher is to reduce element names to their canonical form. The column "TERM" represents the canonical form of element names. The column "SYNONYM" represents semantically equivalent terms for these canonical names.

Element names from a device/web content profile will be used in SQL queries for this MySQL table, in order to determine if the terms used in the profile are contained in the system's ontology.

Description
Table 3.1 below illustrates a section of the rules base.

Table 3.1 : extract from rules base

TERM	SYNONYM
RAM	mem
CPU	Processor
Resolution	ScreenRes
Harddrive	Hard disk
Soundcard	Soundblaster

The rules base can be queried using SQL [25] statements, for example

SELECT TERM FROM RULES_BASE WHERE SYNONYM='mem'

This statement will return the value 'RAM'

New rules can also be inserted into the rules base by using SQL statements, for example

INSERT INTO rulesbase (Term, Synonym) VALUES ('CPU','Processor')

The rules based matcher defines an ontology representing web content and devices It details known synonyms for terms in these domains, truncated/abbreviated versions of the term, similarly spelled terms which are pronounced identically (eg deliverTo, deliver2), and terms which share common substrings (eg representedby, representative)

If the SELECT query formed for any terms from the device/web content profiles returns an empty set, these terms will be sent to the semi-automatic matchers which will attempt to identify any terms in the ontology it may be semantically equivalent to

Input

From the RDF parser the rules based matcher receives element names and element values pairs from web content and device profiles

Output

- **To Compatibility Gauge** Content handling capabilities and resources of device, content types used by web content and hardware requirements for web content These are the canonical forms of element names used in the device and web content profiles

- **To Semi-automatic Matchers:** Element name/value pairs representing web content and device properties, which could not be transformed to a canonical form by the rules based matcher.

3.2.3 Naïve Bayes Classifier

Function

Classify an element name by using a text-classification algorithm to determine what term in the system's ontology it is statistically most likely to be equivalent to.

Description

This module uses the Naïve Bayes algorithm to match an unknown element with an element which it is possibly semantically equivalent to in the system's ontology.

Input

From the rules based matcher: Element name/value pairs representing web content and device properties, which could not be transformed to a canonical form by the rules based matcher.

Output

The Naive Bayes Classifier sends suggestions for the canonical form of element names to the composite matcher.

3.2.4 Neural Net Matcher

Function

Classify an element name by using a neural network to determine what terms in the ontology share similar characteristics.

Description

Uses machine learning algorithm to classify input data by generalising and making inferences from training data.

Input
From the rules based matcher: Element name/value pairs representing web content and device properties, which could not be transformed to a canonical form by the rules based matcher.

Output

The Neural Net Matcher sends suggestions for the canonical form of element names to the composite matcher.

3.2.5 Subsequence Matcher

Function

Identify possible semantic mappings between unknown element names and terms in the system's ontology by detecting common subsequences in terms.

Description
The subsequence matcher looks for terms which share common subsequences, and suggest them to the user as possible semantic matches. For example, it may see that "representedBy" and "representative" share the subsequence "represent", and suggest this pair to the user as a semantic match.

Input
From the rules based matcher: Element name/value pairs representing web content and device properties, which could not be transformed to a canonical form by the rules based matcher.

Output

The Subsequence Matcher sends suggestions for the canonical form of element names to the composite matcher.

3.2.6 Rule Generator

Function

Generate new rules to add to the existing rules base, based on semantic mappings suggested by the neural net, Naïve Bayes, and subsequence matchers.

Description

The rule generator is responsible for generating new rules. It may generate a semantic mapping between attribute X and attribute Y if:

For attribute X, a sufficiently high number of new mappings suggested semi-automatically are to attribute Y. (Exactly what number is configurable).

Input

The Rule Generator receives matches which have been accepted by the user from the composite matcher.

Output

The Rule Generator sends term/synonym pairs to the Rules Based matcher which will be inserted into the rules base to form new rules using SQL INSERT statements.

3.2.7 Composite matcher

Function

Decide which of the three semi-automatic matchers is most likely to be correct, and send the output from this matcher to the rules based matcher and the rule generator.

Description

If the output returned from the semi-automatic matchers is not unanimous, the composite matcher uses the following algorithm for selecting the semi-automatic matcher which is most likely to be correct. If the Naive Bayes matcher returns a clear result, its output is selected. If the Naive Bayes matcher's output is ambiguous (ie multiple categories given probabilities of 1.0, all categories given 0.0), then the composite matcher must choose between the neural net and the subsequence matchers. If the probability associated with the neural net matchers choice is greater than .9, the neural net matcher is selected by the composite matcher, else the subsequence matcher is selected.

Input

From the neural net, subsequence and naive bayes matchers: the element in the rules based matcher's ontology that each of these matchers has calculated an unknown element is most likely to be semantically equivalent to

Output

The output from the semi-automatic matcher that the composite matcher has calculated is most likely to be correct is sent to the rules based matcher and the rule generator.

3.2.8 Compatibility Gauge

Function

Determine if the end user can access web content with a particular device.

Description

The Compatibility Gauge examines the content handling capabilities of the device and the properties of the web content it is trying to access. It analyses the hardware requirements necessary to access the web content (eg screen size, network connection speed) and determines if the hardware specifications supplied by the device meet these requirements. If the device is not capable of accessing the web content, the information about these incompatibilities is sent to the web content provider. The web content provider can use this information to provide alternative web content that is suitable for the user's device.

Input

From the rules based matcher: the canonical form of element names in the device and web content profiles provided by the user, and their associated values.

Output
To the web content provider: A boolean value; true if device is capable of accessing web content; false otherwise. If the device is not capable of accessing the web content, the information about this incompatibility is sent to the web content provider.

4 Implementation of architecture

The application is implemented as a web service. Simply put, a Web service is a web-based application that exposes a programmatic interface using standard, Internet-friendly protocols. The web-services paradigm is highly modular. Web-services and the programs which invoke them are loosely coupled; neither needs to have an in-depth knowledge of how the other works. We can look on web-services as being on-line building blocks for an application. By using web services we can rapidly build new applications or extend the functionalities of existing ones.

4.1 Invoking web services

If someone wanted to create a program which invoked our web service, how would they know the functions exposed by the web service, and the parameters accepted by these functions? The program has to be supplied with a document describing the web service's interface. If we want to create applications which can communicate data to each other autonomously, without human direction, this documentation must be in a standardised, machine readable form.

There does exist a protocol which provides information about web-services in a machine readable format. It is called Web Services Description Language (WSDL) [20]. By using WSDL, it is easy to invoke programs remotely and to allow your programs to be invoked remotely.

4.1.1 Web Services Description Language

A WSDL document defines a web-services interface. It specifies the data which it sends and receives. As The two communicating applications don't have to run on the same platform; WSDL is a platform and application independent definition language; only the message is important. This simplifies the interface between applications.

A WSDL document is a set of definitions. It includes the following parts:

- Types: These are the definitions of the basic data types used in the messages exchanged between the web service and the application invoking the web service.
- Message: An abstract definition of the data being communicated.
- Operation: An abstract definition of the function performed by the service.
- Port: The address of a single web service, defined as a combination of a binding and a network address.
- Port Type: A set of operations performed at a Port.
- Binding: Specification of protocol and data used by a Port Type
- Service: A collection of one or more related ports.

For a sample WSDL document, see Appendix A.

4.1.2 Simple Object Access Protocol (SOAP)

Web services communicate with other applications by using SOAP: Simple Object Access Protocol [22]. SOAP is an XML-derived protocol which uses existing internet transport protocols (such as HTTP) to transmit XML-encoded data. SOAP describes the messages that pass between web services and clients that invoke these services.

SOAP is an easily implemented protocol with widespread industry support. It is a light-weight protocol; its text-based, and therefore operating system and application independent. It uses existing technologies such as HTTP and XML which have gained widespread industry acceptance.

An important feature of SOAP is its ability to enable communication through firewalls. All firewalls pass traffic using port 80, which is used by HTTP. Because SOAP can use HTTP as its transport protocol it is not blocked by firewalls.

Transmissions between applications under the SOAP protocol take the form of messages. A SOAP message is an XML document which consists of an envelope, a header (optional), and a body. *Figure 4.1 shows a simple SOAP message.*

Figure 4.1 : sample SOAP message

```
<SOAP:Envelope>
 xmlns:SOAP='http://schemas.xmlsoap.org/soap/envelope/'
 SOAP:encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'

        <SOAP:Header>
                <Language>
                        English
                </Language>
        </SOAP:Header>
        <SOAP:Body>
                <DoCreditCheck>
                        <ssn>123-456-7890</ssn>
                </DoCreditCheck>
        </SOAP:Body>
</SOAP:Envelope>
```

We will now analyse the components of the above SOAP message (Envelope, Header and Body).

The envelope is the first element in a SOAP message, it encapsulates all the other parts of the message. It identifies the XML document as being a SOAP message, and how it is encoded (that is, how the data is to be serialised). This information is represented as namespace URIs [35].

Below is the envelope element of the SOAP message from figure 4.1, with the Header and the Body of the message removed for the sake of clarity.

```
<SOAP:Envelope>
xmlns:SOAP='http://schemas.xmlsoap.org/soap/envelope/'
SOAP:encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
.
.
.
</SOAP:Envelope>
```

The first namespace URI (**'http://schemas.xmlsoap.org/soap/envelope/'**) specifies the version of the envelope. It identifies the XML document as being a SOAP message. The second URI (**'http://schemas.xmlsoap.org/soap/encoding/'**) specifies the encoding used to serialise the data in the body of the message.

The envelope element may contain a header element. It is optional, and is used to extend the SOAP message syntax. For example it can add features such as authorization or transaction information to the SOAP message, or provide information about the message such as specifying the language of the message.

```
<SOAP:Header>
        <Language>
                English
        </Language>
</SOAP:Header>
```

The Body element must be present in a SOAP message, and it must be an immediate child of the envelope element. It contains the actual message, which is represented as child elements of the body. This could be a method call along with textual representations of the arguments required by the method, or it could be any data represented in an XML format.

The body of the SOAP message in figure 4.1 is a procedure call to a function DoCreditCheck, with one argument (123-456-7890) being passed to the function.

```
<SOAP:Body>
        <DoCreditCheck>
                <ssn>123-456-7890</ssn>
        </DoCreditCheck>
</SOAP:Body>
```

4.1.3 JAX-RPC

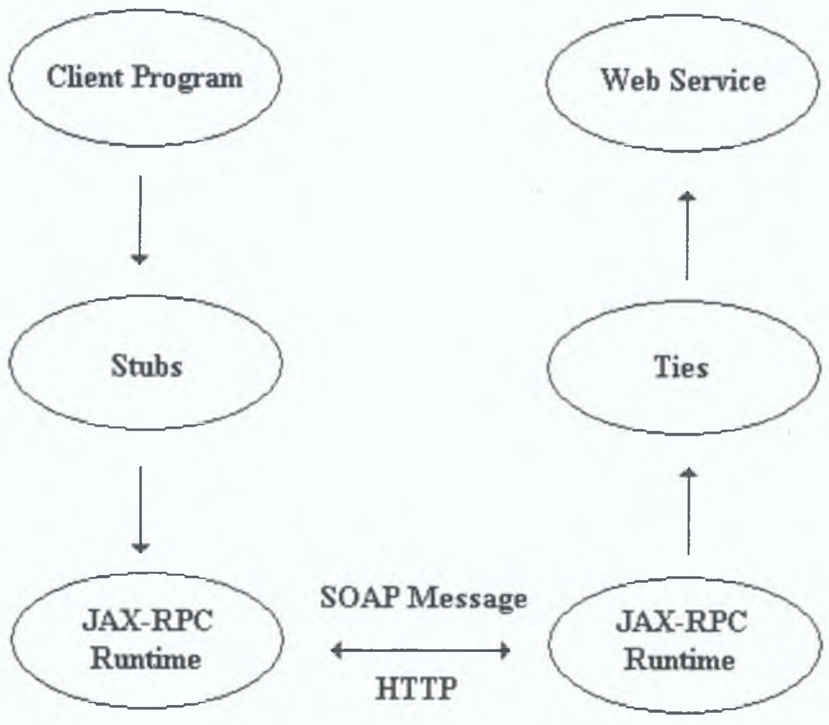
JAX-RPC [23] is the Java API for XML-Based Remote Procedure Call. Remote Procedure Call is a protocol which enables an application running on a computer connected to a network to execute processes on other machines connected to the same network. The JAX-RPC API in the Java Web Services Development Pack [43] is used to enable communication between the Client Service Capability Matcher and the end-user.

JAX-RPC uses SOAP and HTTP to make RPCs over a network. The communication between the client and the server is encoded using SOAP. The SOAP specification defines message structure, encoding rules, and a convention for representing remote procedure calls and responses. HTTP (Hypertext Transfer Protocol) serves as the transport mechanism. Although JAX-RPC relies on complex protocols, the API hides this complexity from the application developer.

Unlike earlier APIs used to make RPCs (for example, RMI [44]), JAX-RPC allows client-server interaction even when the client and the server are implemented using different platforms. JAX-RPC enables a non-Java client to invoke a web service implemented using Java, and vice versa. The interface to a web service is described using WSDL.

Figure 4.2 illustrates the communication exchange between a JAX-RPC client program and a web service.

Figure 4.2: An application using JAX-RPC at runtime



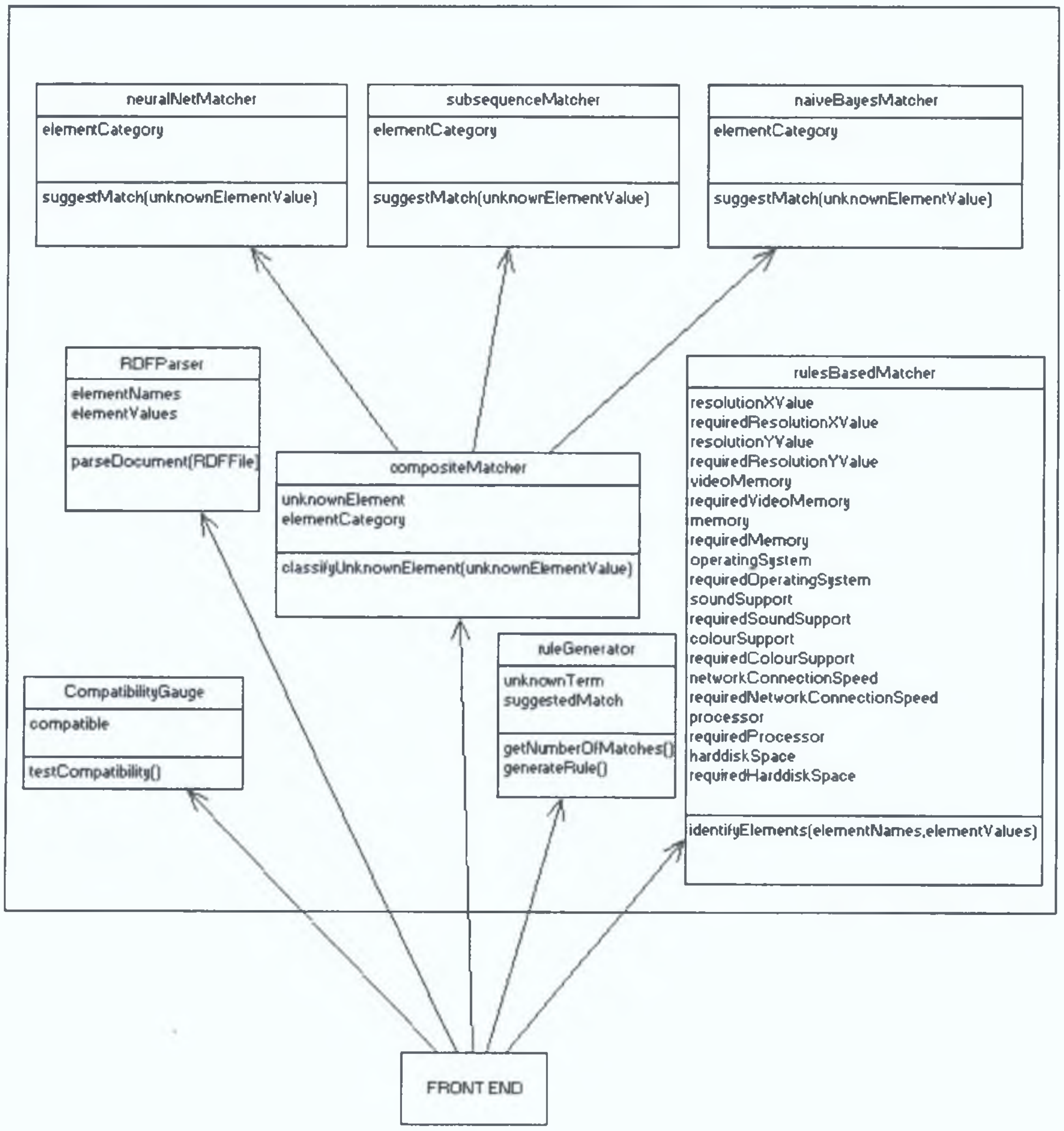
Stubs are local objects that represent the remote procedures. Ties are server-side classes which enable interaction between the client and the web service. This is what happens when a client invokes a function provided by the web service.

- The client calls the method on the stub that represents the remote procedure.
- The stub executes the necessary routines on the JAX-RPC runtime system.
- The runtime system converts this method call into a SOAP message and transmits the message to the server as an HTTP request.
- The server, upon receipt of the SOAP message, invokes the methods on the JAX-RPC runtime. The JAX-RPC runtime converts the SOAP request into a method call.
- The JAX-RPC runtime then calls the appropriate method on the tie object.
- Finally, the tie object calls the method on the implementation of the Web service.
- The response to the RPC call is sent in a SOAP response message as an HTTP response to the client.

4.2 Class diagram of Client Service Capability Matcher

The class diagram below (Figure 4.3) shows the main java classes in our system and the relationships between them.

Figure 4.3: Client Service Capability Matcher Class Diagram



All these classes were implemented using the Java Source Development Kit 2 (version 1.4.0).

4.2.1 Front End

The user interface is a GUI implemented using the Java Foundation Class Swing libraries.Through a series of dialog boxes, the user is asked to provide:

- The URL for the RDF file describing the target web content
- The URL of the target web content
- The URL for the RDF file describing the users device

The Front End is the client for the Client Service Capability Matcher web service. All the other classes are server side web service classes.

4.2.2 RDFParser

The RDF Parser obtains the names of elements and the values of these elements in an RDF document. It uses the Simple API for XML Processing (SAX) [24]. SAX is an event driven mechanism for accessing XML documents. With a SAX parser, events are related to what is currently being read from the XML document, for example:

- Element opening tags
- Element closing tags
- Content of elements
- Parsing errors (in cases where XML document is not well formed)

The SAX API acts like a serial I/O stream. You see the data as it streams in, but you can't go back to an earlier position or leap ahead to a different position. In general, SAX works well when you simply want to read data and have the application act on it. This is perfectly adequate for the requirements of our parser.

The RDF Parser is implemented in the class RDFParser. It is invoked by creating an instance of this class, and calling the parseDocument method of this instantiation. The name of the file to be parsed is supplied as an argument to this method. The method returns a Vector object, containing Strings representing the names of elements in the RDF file supplied as an argument, and the values of these elements.

4.2.3 rulesBasedMatcher

The rules based matcher consults a lookup table, which details pairs of semantically equivalent terms. Below is a selection of semantic mappings from this table:

Table 4.1: Selection of semantic mappings from rules base

TERM	SYNONYM
MB	Megabytes
MB	Meg
MB	Megs
Soundcard	Soundboard
NetworkConnection	ConnectionSpeed
Microphone	Mic
Microphone	Mouthpiece
Speakers	Loudspeakers
Headphones	Phones
Html	Htm
Txt	Text
Txt	Plaintext

The rules based matcher is implemented as a class which sends SQL[25] queries to a MySQL database. Java Database Connectivity[26] drivers are used to send requests to the database and retrieve responses to these requests. For example, if we encountered the term “HarddiskSpace” in an RDF profile, and wanted to consult the rules based matcher to see what this term was equivalent to, we would send the following SQL statement to the database:

```
SELECT TERM FROM TERMS WHERE SYNONYM = ‘HarddiskSpace’
```

4.2.4 naïveBayesMatcher

The Naïve Bayes matcher uses the Naïve Bayes algorithm[27] to classify elements taken from the RDF profile. The Naïve Bayes algorithm is frequently used in text-classification applications [12], [13], [14]. It uses Bayes theorem to calculate the probability that an element/attribute belongs to a particular category given the words its value contains. For example, if the value of an element contains the word “Soundblaster” then the element is highly likely to belong to the category “Sound Card”. The Naïve Bayes algorithm assumes conditional independence between the attributes of the data it is classifying. Conditional independence is the assumption that the effect of one variable on the classification process is independent of all other variables. This is not always true in real-world situations (hence the name, NAÏVE bayes).

Given a hypothesis (H), and an observed fact (D), we calculate the probability of the correctness of the hypothesis given that fact (P(H|D).

For example, in a training set of 40 elements, 23 elements are categorised as representing addresses. 20 of the elements in the training set contain the word “street” in their values. 14 of the elements containing the word “street” in their value are “address” elements.

Given the above, what is the probability that an element is an “address” element if it has the word “street” in its value?

Bayes theorem

P(H|D)=P(D|H)*P(H)/(P(D|H)*P(H)+P(D|H')*(1-P(H)))

Which can be simplified to

P(H|D)=P(D|H)*P(H)/P(D)

H, Hypothesis element is of type "address"

D, Datum, an observed fact element contains word "street"

P(H)=prior probability element is of type address=23/40 = 575

P(D|H)=probability element contains word "street" if it is an address = 14/23 = 6087

P(D)=probability element contains word "street" = 20/40 = 5

P(H|D)= (6087* 575)/ 5 = 7015

4.2.5 subsequenceMatcher

The subsequence matcher detects common subsequences between terms in the application's ontology and unrecognised terms For example, it may suggest "videores" and "videoscreen resolution" to the user as a semantic match

In order to calculate the similarity between two terms, we first calculate the Longest Common Subsequence between the two

Definition Longest Common Subsequence
The Longest Common Subsequence (LCS) of two strings is the longest subsequence of characters (excluding whitespace) that occurs in both strings

For example:

We compare the term “Disk Store” to “Storage”:

The LCS, with a length of 5, is “Store”:

HARDDRIVE TERM
Disk Store

UNKNOWN TERM1
Storage

A problem with using the LCS as a similarity metric between two strings is that the longer an input string is, the more likely it is to contain characters forming a common subsequence with the string we compare it to. If we compare a long input string containing many words to a particular term, it will more than likely be given a higher similarity score than a single word compared to the same term, simply because the likelihood of finding a common substring increases with the length of the input.

In order to prevent comparisons involving long strings getting misleadingly high similarity scores, we calculate a value, the “distribution”, which indicates how “spread out” the LCS is in the unknown term.

For example, we also compare “Disk Store” to “TCP/IP connection requires an Internet account and 28.8 Kbps (or faster) modem”:

The LCS, with a length of 7, is “iskStre”:

HARDDRIVE TERM

D<u>isk</u> <u>Store</u>

UNKNOWN TERM2

"TCP/IP connection requ<u>i</u>re<u>s</u> an Internet account and 28 8 <u>K</u>bps (or fa<u>s</u>t<u>e</u>r<u>)</u> mod<u>e</u>m"_

Our second comparison results in a longer LCS, even though the two terms in the first comparison are more similar

We divide the length of the LCS by the distribution of the LCS in the unknown term, so that an LCS in which the characters do not occur close to each other will result in a lower similarity score than an LCS in which the characters are closer

For example, as seen in the first comparison above, the LCS of "Disk Store" (HARDDRIVE TERM) and "Storage" (UNKNOWN TERM1) is "Store" The first character of the LCS occurs at the first character in UNKNOWN TERM1, and the last character of the LCS occurs at the 7th character in UNKNOWN TERM1 So, the distribution is (7-1)+1=7 (we add the one to prevent divide by zero errors)

Looking at the second comparison, the first character in the LCS occurs at the 23rd character in UNKNOWN TERM2 ("TCP/IP connection requires an Internet account and 28 8 Kbps (or faster) modem") while the last character in the LCS occurs at the 77th character of UNKNOWN TERM2, therefore the distribution is (77-23)+1=55

We also calculate the length of the LCS divided by the length of HARDDRIVE TERM (because just as with the unknown term, the longer HARDDRIVE TERM is, the longer the LCS is also likely to be)

The product of these two values gives the overall similarity score between the unknown term and a term in the system's ontology

$$\frac{\text{LengthOfLCS}}{\text{UNKNOWN TERM Distribution}} * \frac{\text{LengthOfLCS}}{\text{LengthOfKnownTerm}}$$

In the above examples, the LCS between HARDDRIVE TERM and UNKNOWN TERM1 is 5 The distribution of the LCS in UNKNOWN TERM1 is 7 The length of HARDDRIVE TERM is 10 So, the similarity score between HARDDRIVE TERM and UNKNOWN TERM1 is 5/7 * 5/10 =25/70= 35714

The LCS between HARDDRIVE TERM and UNKNOWN TERM2 is 7 The distribution of the LCS in UNKNOWN TERM2 is 55 The similarity score between HARDDRIVE TERM and UNKNOWN TERM2 is 5/55 * 5/10 =25/550= 04545, which is much lower than the similarity score between HARDDRIVE TERM and UNKNOWN TERM1 These results reflect that the two terms in the first comparison are more similar than those in the second comparison

During testing, the subsequence matcher proved itself to be the most effective of the semi-automatic matchers in classifying elements with lengthy textual values (see "Results", chapter 6)

Table 4 2 shows the accuracy of the semi-automatic matchers when used to categorise the 48 elements with the longest values in the test data (10-12 words, eg "150 MB free hard disk space plus space for saved games", "256 MB RAM or higher (more memory normally results in improved performance)"

Table 4.2

NEURAL NET	NAÏVE BAYES	SUBSEQUENCE MATCHER
34/48 (71%)	28/48 (58%)	40/48 (83%)

The subsequence matcher is clearly the most effective matcher when categorising elements with lengthy textual values.

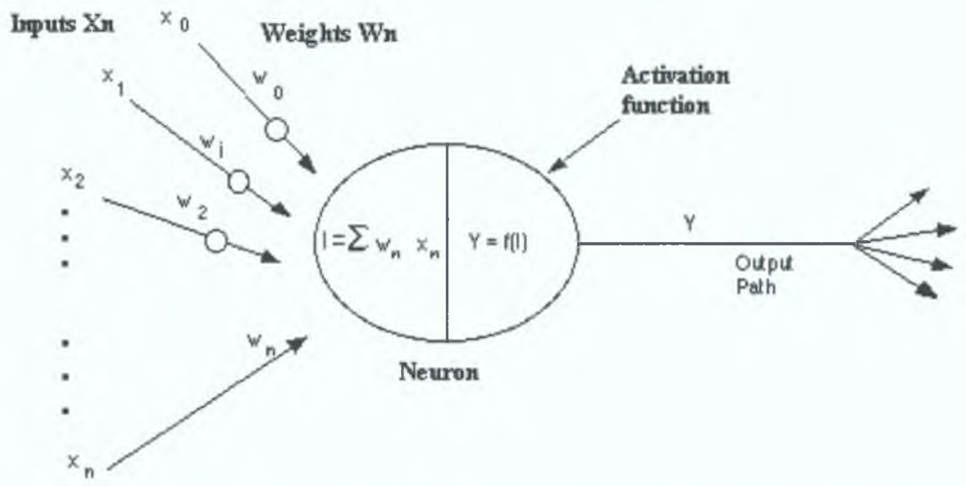
4.2.6 neuralNetMatcher

Neural networks[27] are statistical models of real-world systems. Neural networks are based on biological concepts, modelled on how neurons process information in the human brain. They can classify data by recognising patterns in the data.

The basic element in an artificial neural network is a neuron. This corresponds to a neuron in a biological brain. It receives input from other neurons, or from an external source. There are three different types of neuron:

Input neurons: receive data from external source
Output neurons: send data to an external source
Hidden neurons: perform intermediate calculations between input and output neurons

Figure 4.4: A neuron



Neurons are linked by connections. Each connection has a weight associated with it. The input to a neuron via a particular connection is multiplied by the weight associated with that connection. These values are then summed and passed into an "activation function" which calculates the value output from this node. An example of an activation function is the "step function" which returns 1 if the summed input is above a certain threshold, else it returns 0 (see figure 4.5). Another activation function is the sigmoid function, which returns continuous output in the interval 0 to 1 (see figure 4.6). There are many other activation functions. Each function is suited to particular scenarios, therefore the choice of activation function depends on the characteristics of the problem the neural net is attempting to solve.

Figure 4.5 : Step Function

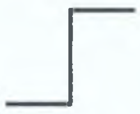
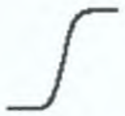
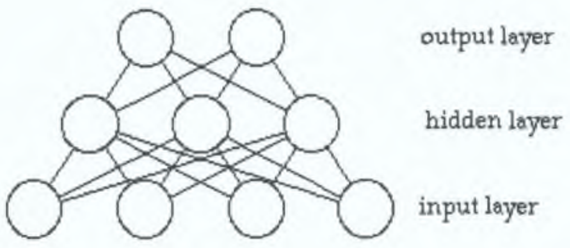


Figure 4.6 : Sigmoid function



The most commonly used neural network model is the Multi-Layer Perceptron (MLP). Figure 4.7 illustrates an example of an MLP. The MLP is arranged in layers of nodes so that values from the input neurons are propagated to the hidden neurons. The hidden neurons may be arranged in more than one layer. The output from each layer of hidden neurons is propagated to the next until it reaches the layer of output neurons.

Figure 4.7 : An MLP with one hidden layer.



How does the neural net learn to classify input data?

1. Build a network with the chosen number of input, hidden, and output units.
2. Initialise all the weights to low random values.

REPEAT:

3. Choose a single training pair at random
4. Copy the input pattern onto the input layer
5. Calculate the value produced at the output layer
6. Calculate the error between the obtained output and the desired output
7. Back propagate the summed product of the weights and errors in the output layer in order to calculate the error on the hidden units
8. Update the weights into each unit according to the error on that unit, the output from the unit below and the learning parameters
9. Update the connection weight values to the output layer by using this equation

UNTIL the error is sufficiently low or the network settles

Our applıcatıon uses neural networks to perform a classıficatıon task The ınput ıs a descrıptıon of the devıce/web content property to be recognısed, and the output ıs the class to whıch thıs property belongs (eg RAM/CPU/Vıdeo Card/etc)

The neural net applıcatıon ıdentıfies ınput as beıng most lıkely to belong to one of the followıng categorıes

- Soundcard
- Vıdeocard
- Resolutıon
- Colour
- RAM
- VıdeoRAM
- Operatıng System
- Network Connectıon
- Harddrıve
- CPU

For the sake of sımplıcıty we lımıted the number of categorıes to 10, thıs can be expanded ıf requıred

The neural net ın our applıcatıon has three layers
59 ınput nodes
100 nodes ın a hıdden layer
10 output nodes
The neural net ın our applıcatıon, wıth 59 ınputs and 10 outputs, represents a 59-dımensıonal dataspace Any ınput to the neural net represents a poınt ın thıs dataspace

The neural net has been trained to recognise to which category every datapoint is most likely to belong to.

A useful property of neural nets is their ability to generalise, ie the outputs of the neural net approximate target values for inputs that are not in the training set. The neural net can use its generalisation abilities to "fill in" spaces in the training dataset.

In order to implement our neural net application, we used the Java Object Oriented Neural Engine[28] (JOONE). This is an open-source API and GUI editor that provides the components required to create a neural net application.

How does the neural net classify textual data?
The input to our neural network matcher requires 59 numbers. The raw data is a text string however. The input must be transformed into a numerical form suitable for the neural net. We will now examine the steps involved in this transformation.

If we want to represent the string "90 MB of available space required on system drive" in a format suitable for the neural net matcher, we follow this procedure:

1. Does the term megabyte (or a synonym of megabyte) occur in the string?

    Yes it does, and the value associated with it is 90. The first 10 input nodes are used to represent this number in binary form. (More nodes can be added if we want to represent larger numbers, but in order to minimise the number of input nodes, for now we only have 10). Therefore, the value for the first 10 nodes is 0,1,0,1,1,0,1,0,0, and 0.

    Node 1 is the Least Significant Bit, while node 10 is the Most Significant Bit.

2. Does the term gigabyte (or a synonym of gigabyte) occur in the string?

    No it does not, so the value for nodes 11 to 28 is set to 0. If the input string contained "20GB", these nodes would contain the binary representation of the number 20,000 (the number is multiplied by 1,000 because quantities of disk space expressed in gigabytes are frequently decimals, eg 4.2 gigs).

3 We now compute the similarity score using the modified LCS metric (as used by the subsequence matcher) between the input and each of the following 29 terms:

    Audio, Sound, Video, Screen Resolution, Res, Screen Res, VideoRes, VideoResolution, Resolution, Colours, Color, Colourdepth, Colourcapability, Coloursupport, Colour, Megabitspersecond, Mbps, Megahertz, Mhz, Ghz, HarddiskDrive, Storage, FixedStorage, Massstorage, Diskstorage, Diskspace, Harddisk, Hard-disk, Hard-drive, Harddrive.

    These scores are the values placed into nodes 29 to 58.

4. The final node contains 1 if any of the following terms are present in the input data, else it is set to 0:

    "Windows", "Win95", "Win98", "Solaris", "Unix", "Linux".

The numerical input to the neural net can be represented as a graph, with units on the x-axis representing input nodes and units on the y-axis representing values for these nodes.

The graph representing the input node values derived from the string "90 MB of available harddisk space required" is as below (Figure 4.8).

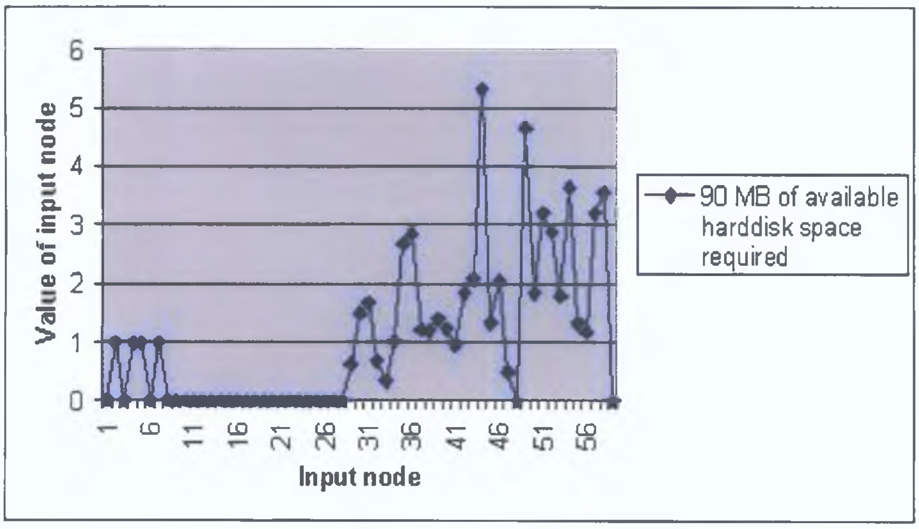


Figure 4.8 : Input node values derived from the string "90 MB of available harddisk space required"

The more similar two text strings are, the more the graphs representing them will resemble each other. If we transform data representing similar strings (ie "Min 40MB disk space", "200 mbytes hard-disk", "80MB hardrive space", etc), the graphs will broadly resemble each other (though this might be true only for certain sections of the graph). This is demonstrated in figures 4.9 to 4.11 where the input node values derived from the strings "Windows 98/ME/2000SP2/XP", "High Colour", and "60 MB of available hrd-dsk space" are overlaid onto figure 4.8.

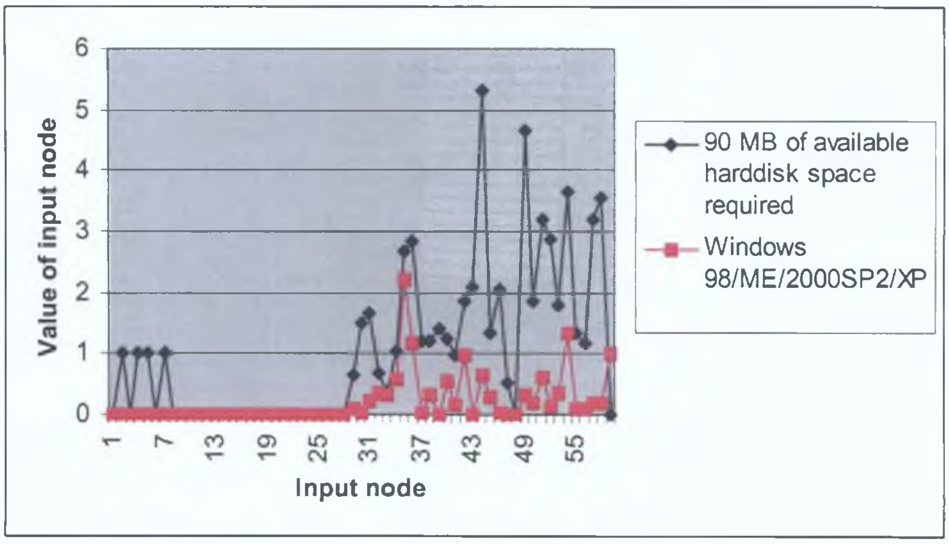


Figure 4.9 : Input node values derived from the string "Windows 98/ME/2000SP2/XP"

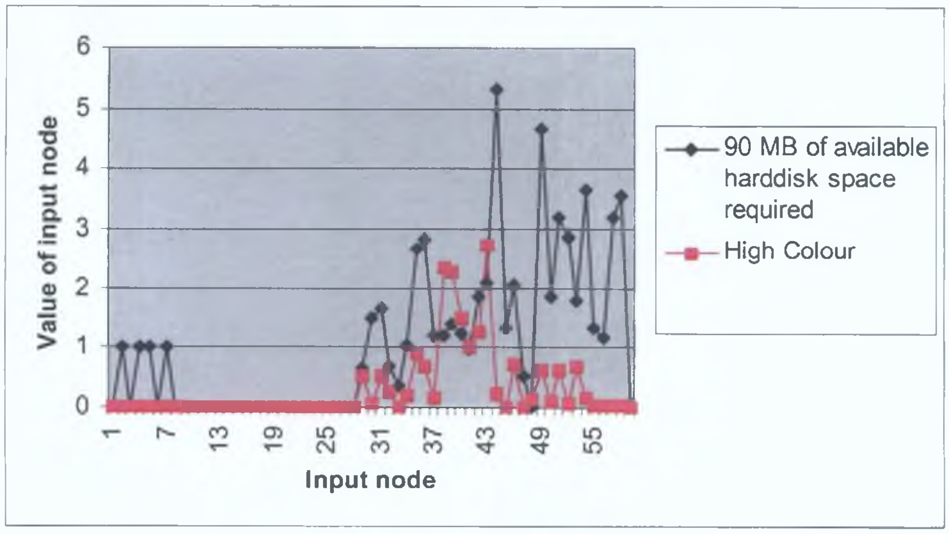


Figure 4.10 : Input node values derived from the string "High Colour"

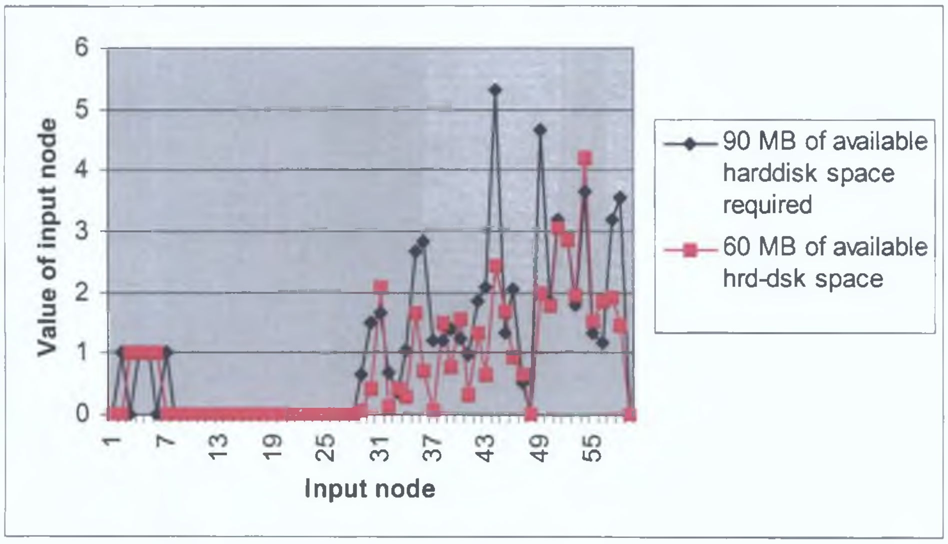


Figure 4.11: Input node values derived from the string "60 MB of available hrd-dsk space"

Clearly, the chart in which the overlaid graph line most closely approximates figure 4.8 is figure 4.11. This example shows that similar text strings will result in similar inputs to the neural net. The neural net can learn to identify common patterns between similar strings.

The neural net matcher generates ten probability scores, indicating the probability that the input belongs to a particular category. Table 4.3 gives an example of such scores generated by the neural net matcher, alongside the category with which each score is associated:

Table 4 3 · Neural net output

Probabılıty Score	Category
4 867118490494595E-4	Soundcard
8 837659506556217E-5	Vıdeocard
1 7031400662453158E-9	Resolutıon
0 9056907276388094	Colour
4 000241468415735E-5	RAM
2 959698019949181E-7	VıdeoRAM
3 2856512031200196E-4	Operatıng System
1 55842706380494E-8	Network Connectıon
3 8890873660489465E-9	Harddnve
2 3120086704091745E-6	CPU

The neural network selects the hıghest score and returns as output the category assocıated wıth that score, ındıcatıng that ıt ıs the most lıkely category that the ınput belongs to If the neural net generated the scores ın Table 4 3, ıt would return "Colour (ıts value, 0 9056907276388094, ıs the hıghest)

If we traın a neural net by presentıng ıt wıth ınput representıng data such as "200 mbytes hard-dısk" and "80MB hardnve space' that produce the output *harddrıve*, ıt wıll eventually learn to recognıse ınputs sımılar to these as representıng harddnve space The neural net wıll learn to recognıse patterns ın the ınput nodes that appear when partıcular words or phrases (for example, dıskspace, Harddnve, MB) are present ın the raw text

The neural net can learn to recognıse the sımılanty scores resultıng from the presence of words that frequently occur ın the traınıng data Thus, the neural net can learn to ıdentıfy words in the input data that strongly indicate that the data belongs to a particular category. The neural net matcher can recognise input based on how similar it is to 29 terms related to the domain of device capabilities/requirements. The neural net matcher can be trained to identify words which indicate that the input belongs to a particular category, even if these words are do not belong to the set of 29 terms which the input data is compared to. The values assigned to nodes 29-58 in the input layer of the neural net describes the input string in a manner not dissimilar to that employed by puzzles such as the following:

My first is in tea but not in leaf
My second is in teapot and also in teeth
My third is in caddy but not in cosy
My fourth is in cup but not in rosy
My fifth is in herbal and also in health
My sixth is in peppermint and always in wealth
My last is in drink, so what can I be?
I'm there in a classroom, do you listen to me?

(The answer is teacher).

The word "teacher" is described in terms of how similar it is to other words (ie what words it shares a letter with and what words it doesn't).

Similarly, in our application, a phrase such as "High Colour" is described in terms of how similar it is to the terms "audio", "screen res", "colour", "harddiskdrive", amongst others.

During the time we spent testing the neural network, it was found that if the neural network was presented with inputs representing the string "Pentium Class Processor" it would successfully classify this data as being of type CPU, based purely on how similar it was to terms like audio, screen res, colour, etc. Similarity scores are not computed for how similar the input data is to "Pentium", "Class", "Processor ', or any combination of these words

We don't have to tell the neural network explicitly "If the input data contains the words 'Pentium Class Processor', this input is of type 'CPU'", the neural network is able to infer this from the training data

4.2 7 compositeMatcher

When the system encounters an element name that is not present in its ontology, it attempts to identify the semantic meaning of the element name based on the value of the element The composite matcher accepts as input the value of an element that the rules based matcher was unable to classify (unknownElementValue) The method classifyUnknownElement passes unknownElementValue to each of the three classes implementing the semi-automatic matchers (neuralNetMatcher, subsequenceMatcher, naiveBayesMatcher)

Each of these matchers returns what they calculate is the most likely term in the rules base this unknown term is semantically equivalent to

The individual matchers often return conflicting matches One matcher could indicate an element represents CPU speed, another could indicate it represents network connection speed, and another could indicate it represents RAM

When the individual matchers produce conflicting results, the composite matcher must decide which of them is most likely to be correct We designed an algorithm which calculates which of the semi-automatic matchers is most likely to be correct, and selects the output of this matcher as the final result of the semi-automatic matching process

The algorithm works as follows

The naive bayes matcher calculates a probability score between 0 and 1 for each of the categories of data it recognises This indicates the probability that the input data belongs to each of those categories If one of these probability scores is higher than every other probability score calculated by the Naive Bayes Matcher, the composite matcher selects the Naive Bayes matcher as the most likely of the matchers to be correct

There are three scenarios where the naive bayes matcher will be unable to provide a clear answer The first scenario is where the input data contains no terms that appear in the matchers corpus In this case, the matcher will return a probability of 0 for every category The second is where the input data contains words which strongly indicate that element belongs to multiple categories "1Ghz processor required for Microsoft Windows" is such an example If the terms "1Ghz" and "processor" only occur in the matcher's corpus labelled as belonging to the category "CPU", and "Microsoft" and "Windows" only occur in the corpus labelled as belonging to the category "Operating System", for both the categories "CPU" and "Operating System" the matcher will return probabilities of 1 The final scenario is where two categories receive an equal probability by coincidence In all these scenarios the result returned by the Naive Bayes matcher will not allow us to categorise the input data, and we must turn to the other two matchers But how do we decide which result from these other matchers is most likely to be correct?

When the neural net matcher categorises unknown data, it returns the category that data is most likely to belong to and a probability score indicating the likelihood of the data belonging to this category The purple line in graph 5 shows the accuracy of the neural net matcher for mappings with probabilities above particular thresholds For example, 72% of semantic mappings with associated probabilities higher than 0 1 are correct 86% of semantic mappings with associated probabilities higher than 0 95 are correct

The blue line in figure 4.12 shows the accuracy of the subsequence matcher on the same data. For example, it has an 86% accuracy rate on the data that the neural net matcher classifies with a probability greater than 0.1. It has an 83% accuracy rate on the data that the neural net matcher classifies with a probability greater than 0.95.

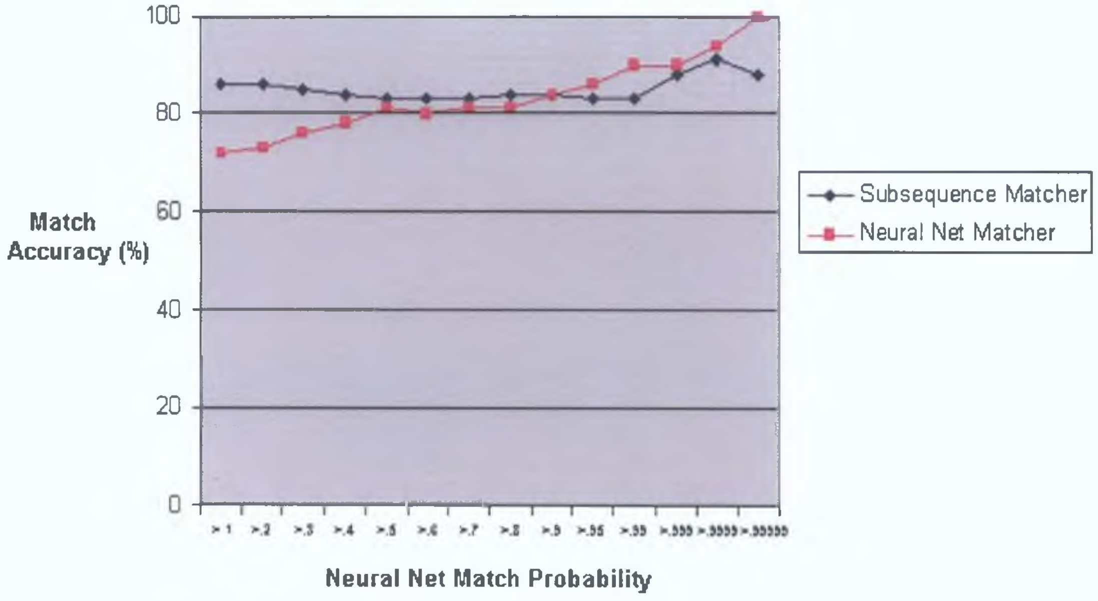


Figure 4.12 : Match accuracy against neural net probability

Figure 4.12 clearly shows that the higher the probability associated with a semantic mapping made by the neural net, the more likely it is to be correct. It shows that if the neural net matcher classifies data with a probability greater than 0.9, it is more likely to be correct than the subsequence matcher.

Thus, if the neural net returns a probability greater than .9 for input data belonging to a particular category, the composite matcher selects this as the matcher most likely to be correct. A high probability such as this strongly indicates that the neural net matcher has correctly classified the input data.

If the neural net does not return a probabilıty greater than 9 for the ınput data belongıng to any category, the composıte matcher selects the output from the subsequence matcher as beıng the most lıkely to be correct

4 2.8 ruleGenerator

The Rule Generator creates new mappıngs ın the rules base based on user ınteractıon wıth the semı-automatıc matchers The rules base can thus expand to ınclude prevıously unknown terms

If the Composıte Matcher consıstently suggests a partıcular semantıc mappıng to end-users (ıe screenRes=Screen Resolutıon), after thıs mappıng has been accepted by end-users a certaın amount of tımes (eg 3), the rule “screenRes=Screen Resolutıon’ can be added to the rules base When the term “screenRes” ıs encountered from then on, the system wıll automatıcally recognıse ıt as representıng Screen Resolutıon, and wıll not ınvoke the semı-automatıc matchers ın order to classıfy ıt

The MySQL database used to ımplement the rules base also contaıns a table “Matches” whıch contaıns every match made by the composıte matcher When a match ıs suggested by the composıte matcher, the system checks the “Match” table to see how many tımes thıs match has prevıously occurred It uses thıs SQL statement

```
SELECT count(*) from matches where term=<suggestedMatch> AND
MatchedWıth=<unknownTerm>
```

An example of thıs statement ın practıce ıs

```
SELECT count(*) from matches where term=’Resolutıon’ AND
MatchedWıth=’ScreenRes’
```

If the number of occurrences of thıs match ıs less than 3 we enter thıs match ınto the Match table usıng thıs SQL statement

```
ınsert ınto matches (Term,MatchedWıth) values ('Resolutıon','ScreenRes')
```

If the number of occurrences of this match is greater than 3 we add a new rule to the table representing the rules base, indicating a semantic mapping between the unknown term and the suggested term

```
insert into rulesBase (term,synonym) values ('Resolution','ScreenRes')
```

The number of times a match has to be suggested before it is added to the rules base was set to 3 for convenience during the testing of the system In practice, if this application had many users this number would have to be higher

4.3 Operation of Client Service Capability Matcher

The W3C Device Independence activity [1] envisions that web content that can be accessed on any device, regardless of the hardware/software capabilities of that device The range of devices that currently access the internet vary enormously with

regards to their hardware/software capabilities For example, devices can vary with regards to

- display sizes and resolution
- sound capabilities
- persistent storage available
- memory
- input capabilities
- network connection speed
- markup/scripting languages supported

These differences in device capability are typical of the issues that must be considered by designers of web content who wish their material to be device independent

The W3C's Device Independence activity [1] recommends that web-content be tailored to the device used to access it An application which tailors web content to the capabilities of a particular device must access information detailing the attributes of both the access mechanism and the web content

Devices and web content can be described using Resource Description Framework (RDF) [5] RDF is intended to describe online resources (for example PCs, applications, data) However, RDF does not define a standardised vocabulary (or ontology) for describing these resources The ontology used in an RDF profile to describe a PC's hardware and software capabilities may differ from the vocabulary that a web site personalisation application may use to represent the properties of a device trying to access the web site content The terms used in the profile describing the device must be translated into the semantically equivalent terms in the personalisation application's ontology

One method of performing this translation is to use a rules based schema matcher A rules based schema matcher consists of a lookup table indicating semantically equivalent terms, for example

Mouse=MouseDevice

Memory=Mem

Keyboard=keybrd

This is effective when we are mediating between only two sources A rules base can be constructed manually Each term in one ontology can have a rule matching it to a semantically equivalent term in the other ontology (if one exists) However, this approach does not scale If we have to mediate between a large number of ontologies (as in the case of mediating between the ontology of a web content personalisation application and the ontologies used by every device that accesses the web content) it is not feasible to manually construct a rules base mapping semantically equivalent terms between the personalisation application's ontology and this large number of device ontologies.

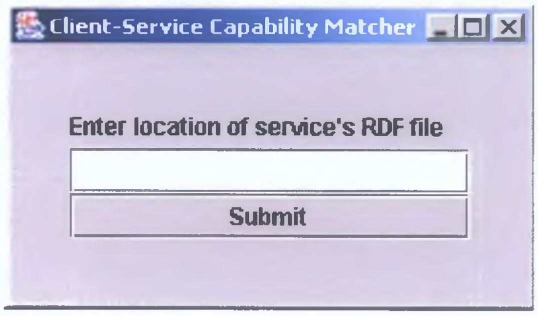
However, we can use heuristic algorithms to semi-automatically create mappings between ontologies. Applications using these algorithms can suggest possible semantic mappings which a human can accept or reject. This removes much of the tedium of mapping between ontologies.

The Client Service Capability Matcher is an application which uses both a rules based matcher and 3 matchers which uses heuristic algorithms (a naive bayes matcher, a substring matcher, and a neural net matcher) to mediate between ontologies. The Client Service Capability Matcher is a web service which can be invoked from a user's device via a GUI .

The application first must be supplied with the URL of the RDF file describing the requirements for accessing target web content

(eg www.fifa.com/worldCupFinalVideoStreamHighResolution.rdf). This RDF file will describe the hardware necessary to access the web content (eg screen size, sound support, etc.).

Figure 4.13 : Dialog box from Client Service Capability Matcher application


Client-Service Capability Matcher
Enter location of service's RDF file
Submit

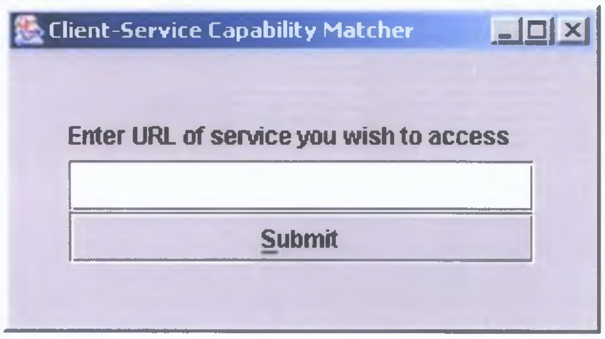
Below ıs an example of an RDF profile descrıbıng the requırements for accessıng a web servıce provıdıng vıdeostreamed output

Fıgure 4 14 RDF profile descrıbıng requırements for vıdeostream

```
<rdf RDF

  xmlns rdf="http //www w3 org/1999/02/22-rdf-syntax-ns#" >

            <rdf Descrıptıon about="AUDIOSTREAM">

            <Sound>Soundblaster</Sound>

            <RAM>128</RAM>

            <NetworkSpeed>100bps</NetworkSpeed>

            <CPU>486DX66</CPU>

            <FıleFormat>mpeg</FıleFormat>

            </rdf Descrıptıon>

</rdf RDF>
```

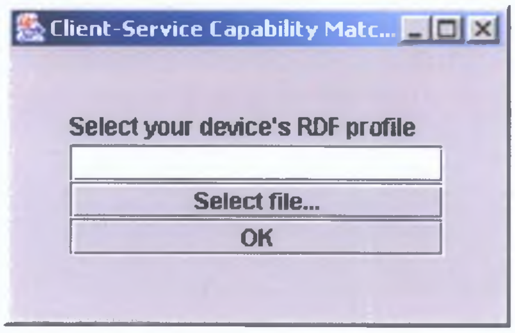
Next, the URL of the web content that the devıce wıshes to access must be provıded (eg www fifa com/worldCupFınalVıdeoStreamHıghResolutıon ram)

Figure 4.15 : Dialog box from Client Service Capability Matcher application


Client-Service Capability Matcher
Enter URL of service you wish to access
Submit

Then, the location of the RDF file describing the access device must be supplied: Eg, c:\metadata\device_profile.rdf

Figure 4.16 : Dialog box from Client Service Capability Matcher application


Client-Service Capability Matc...
Select your device's RDF profile
Select file...
OK

Here ıs an example of a profile descrıbıng an access devıce

Fıgure 4 17 RDF profile descrıbıng devıce

```
<rdf.RDF xmlns rdf="http //www w3 org/1999/02/22-rdf-syntax-ns#" >

                <capabılıtıes>

                <resolutıon>800x600</resolutıon>

                <Colour>16 mıllıon</Colour>

                <VıdeoRAM>120Meg</VıdeoRAM>

                <RAM>1</RAM>

                <Fıleformat><rdf Bag>

                        <rdf lı resource="mpeg"/>

                        <rdf lı resource="word"/>

                        </rdf Bag>

                </Fıleformat>

                <Sound>Soundblaster</Sound>

                <OperatıngSystem>Wındows2000</OperatıngSystem>

                <NetworkSpeedx>100bps</NetworkSpeedx>

                <HardDısk>60</HardDısk>

                <CPU>800</CPU>

                </capabılıtıes>

         </rdf RDF>
```

The RDF Parser extracts the names of elements and attributes from these profiles It must now determine what term in the Client Service Capability Matcher's ontology each of these names are equivalent to

The application first consults its rule base This is a lookup table indicating which terms in the application's ontology are semantically equivalent The lookup table contains mappings such as RAM=Memory, CPU=Processor, CPU=ChipSpeed, etc If the application encounters an element with the name ChipSpeed, by consulting its lookup table it can determine that this element represents CPU

If the application cannot find the term in the rules base, the semi-automatic matchers must determine which term it is most likely to be semantically equivalent to There are three semi-automatic matchers a neural net matcher, a Naive Bayes matcher, and a subsequence matcher The unknown term is input to each semi-automatic matcher, and a composite matcher selects the semi-automatic matcher which is most likely to be correct

The Rule Generator creates new mappings in the rules base based on user interaction with the semi-automatic matchers The rules base can thus be expanded to include mappings which were generated semi-automatically This is a novel feature in our application Unlike other schema matching applications, it can add new rules to the rules base based on interaction with users

The Client Service Capability Matcher will translate the terms used in the profiles describing the device and the web content it is trying to access to their equivalents in its ontology The Client Service Capability Matcher will now be able to directly compare the devices attributes with the requirements for the web content it is trying to access The web content provider can be informed of any ways in which the device is incompatible with the web content

If the device is not capable of accessing web content, a personalisation application can return a URL for web content that is suitable for the device's capabilities For example, if the device's network connection is not fast enough to access a high quality videostream (www fifa com/worldCupFinalVideoStreamHighResolution ram), the personalisation application can return a URL to a lower quality videostream that requires less bandwidth (eg www fifa com/worldCupFinalVideoStream**Low**Resolution ram) The web content provider can maintain a number of different versions of the web content, and the output from the Client Service Capability Matcher will enable the personaliation application to match the device to the appropriate version Customised versions of web content could also be generated dynamically by the personalisation application The URL returned by the Client Service Capability Matcher to the user's device would point to these dynamically generated pages

5. Evaluation of architecture and implementation

5 1 Naive Bayes Matcher

The naive bayes classifier used in our architecture has also proved to be quite effective in classifying device and software attributes taken from profiles from a wide range of sources While testing our naive bayes matcher, it correctly identified the category to which an unknown element belonged in 72% of cases There are however some scenarios in which the naive bayes algorithm is not effective

Consider the scenario where we wish to classify an element with a value of "20 MB required for Windows XP"

The Naive Bayes matcher is not be able to classify this text It considers it equally likely to belong to the category "Operating System" as to the category "Harddisk space", because the text contains terms which indicate that it belongs to each of these categories

The matcher fails in this case because of its assumption of conditional independence

The variables in this example are as follows

Variable 1 Does text contain "MB"? (TRUE/FALSE)
Variable 2 Does text contain "Windows XP"? (TRUE/FALSE)

The outcomes that the naive bayes matcher can predict are

Outcome 1 Text belongs to "Harddisk space" category
Outcome 2 Text belongs to "Operating system" category

In a real world scenario, Variable 1 and Variable 2 **are not conditionally independent** If Variable 1 is TRUE, then the value of Variable 2 will have much less effect on the overall probability calculation

The naive bayes matcher indicates that the text is just as likely to belong to the category "operating system" as it is to the "harddisk space" category, even though the presence of the letters "MB" in the text should lower the probability that the text is of type "operating system"

Also, if the data we wish to classify contains no words that appear in the Naive Bayes matcher's corpus of terms, the matcher will return a probability of zero for the data belonging to any of the categories it recognises These can raise problems, such as the matcher being unable to identify a term because it uses an unrecognised spelling (eg color instead of colour) or if an unrecognised abbreviation is used (eg screenRes)

In tests, the accuracy of the Naive Bayes matcher fell from 88% when categorising data that contained data that contained clear indicators that it belonged to a particular category (eg "64MB") to 60% when trying to identify data which contains terms which strongly indicate that element belongs to multiple categories ("19 Megabytes for the Windows version")

The Naive Bayes Matcher also performs poorly with lengthy data (eg "Hardware accelerated D3D compatible 4MB video card with DirectDraw(TM) compatible driver") When tested with lengthy data, the accuracy of the Naive Bayes Matcher was 58%

5.2 Subsequence Matcher

The subsequence matcher uses a string similarity metric based on the longest common subsequence (LCS) metric

A problem with using the LCS as a similarity metric is that the longer a string is, the more likely it is to contain characters forming a common subsequence with another string we compare it to

However, we have devised a modifed LCS metric in our application which takes into account the length of the LCS relative to the length of the strings being compared, the longer these strings are compared to the LCS, the lower the similarity score generated by the subsequence matcher

As an example of the subsequence matcher in use, during tests (as detailed in Appendix B), we tried to classify an element with the value "1 GB or more of available storage" The substring matcher determined that the string "1 GB or more of available storage" was most similar to the string "10 GB or more" in the Client Service Capability Matcher's ontology The string "10 GB or more" belongs to the category "Harddrive", so the subsequence matcher suggests that the element with the value "1 GB or more of available storage" is most likely to represent harddrive capacity

Neither the Neural Net nor the Naive Bayes matcher classified this element correctly The Neural Net Matcher classified it as being CPU speed, and the Naive Bayes matcher did not return a clear clear match (it calculated that it has a probability of 1 0 of belonging to both the RAM and harddrive categories)

The subsequence matcher is the most accurate of the semi-automatic matchers for categorising lengthy data When tested against data which contained words strongly indicating that element belongs to a particular category, and 6-10 other words, the subsequence matcher had an accuracy of 83%, compared to 71% for the Neural Net matcher and 58% for the Naive Bayes matcher Our modified LCS algorithm, which takes into account the length of the unknown data, makes it effective in identifying substrings in data which strongly indicate the category to which data belongs

The subsequence matcher performed particularly poorly however when used to classify data which contained words which strongly indicated that the element belonged to multiple categories It was the poorest of the three semi-automatic matchers in classifying data belonging to this category, with an accuracy of 30% When the matcher has to classify data such as "32 MB required for Windows 98 Operating System", because "Windows 98 Operating System" is longer than "32 MB", the subsequence matcher mistakenly categorises this as "Operating System", instead of "RAM required"

5.3 Neural Net Matcher

Neural networks were designed with the classification of numerical data in mind As much of the data in our domain of interest is textual, the neural network matcher is not as accurate as the Naive Bayes and subsequence matchers (both of which perform strongly on textual data)

However, the neural net matcher can match elements which the Naive Bayes and subsequence matchers cannot Its inclusion in the composite matcher thus allows the system to categorise a wider range of data

5.3.2 Neural net matcher's generalisation capabilities

Table 5 1 represents a portion of a training dataset It shows how often amounts of megabytes occur in the context of RAM and harddisk space There are some amounts of megabytes for which there are no occurrences in the context of either RAM or Harddisk The Naive Bayes matcher will have difficulty classifying these amounts as being either RAM or harddrive, there is nothing in this training data which suggests which category they belong to

TABLE 5 1 Portion of training dataset

Megabytes	Harddrive	RAM
16MB	4	20
17MB	6	0
18MB	5	1
19MB	8	0
20MB	0	3
21MB	0	0
22MB	0	0
23MB	0	0
24MB	4	9
25MB	2	0
26MB	5	0
27MB	0	0
28MB	0	0
29MB	0	0
30MB	7	0
31MB	2	0
32MB	3	24

Figure 5 1 shows the output returned by the naive bayes matcher trained with this dataset, indicating the probability that the terms along the X-axis are of type Harddrive or of type RAM It can be seen that the naive bayes matcher returns values of 0 for inputs in the ranges 21-23MB and 27-29MB It is unable to determine what category these values of megabytes are most likely to belong to

The neural net doesn't have this problem, it is able to generalise, and can produce a function which "fills in" the holes in the dataset

Figure 5 1 output from naive bayes matcher on dataset from table 5 1

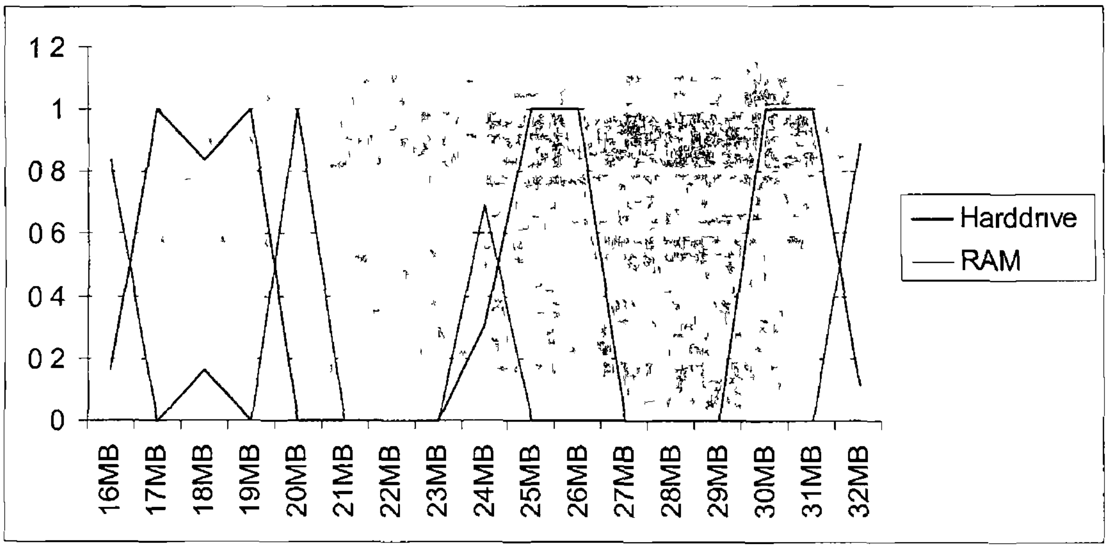
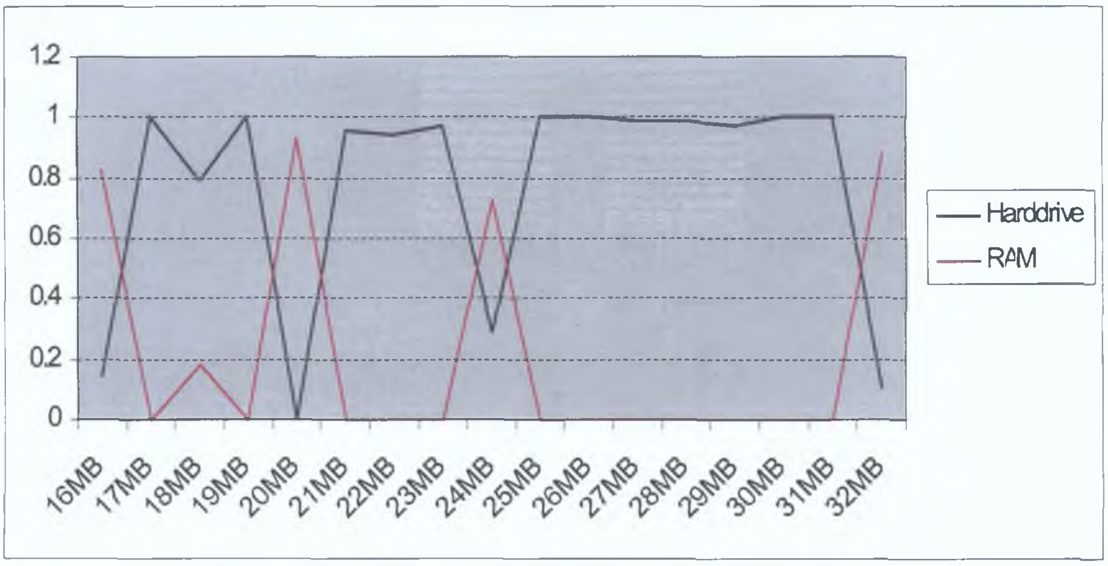


Figure 5 2 shows the output returned by the neural network trained with this dataset It doesn't return values of 0 for both RAM and harddrive when asked to classify 21-23MB and 27-29MB The neural net is able to generalise from the training data and can "fill the gaps" in the data to create an approximation of the desired function

Figure 5.2 : output from naive bayes matcher on dataset from table 5.1



In addition to their ability to generalise, neural networks can perform matches in situations where a naïve bayes or a subsequence matching approach would fail. For instance, during testing the Naïve Bayes and subsequence matchers incorrectly classified the text “65 MB required for Windows 2000 compatible version”.

The Naïve Bayes matcher considered it equally likely to belong to the category “Operating System” as to the category “Harddisk space”, because the text contains terms which indicate that it belongs to each of these categories.

The subsequence matcher also classifies the element as belonging to the “Operating System” category.

However, the neural net matcher successfully classified this text as “Harddisk space”, because it has learned that if text contains “65 MB” it is most likely to belong to this category, regardless of what other text appears in the input string.

5.4 Composite Matcher

We use a composite matcher in order to classify as broad a range of element values as possible For example, if we want to classify x

<x>Soundblaster</x>

The Naive Bayes matcher would easily identify x as representing "Soundcard" The Naive Bayes algorithm works best when the data it is trying to identify contains words that give a strong indication of the category the data belongs to

But, if we want to classify

<x>Sndblster</x>

The Naive Bayes matcher may have difficulty classifying this If the term "Sndblster" does not appear in the Naive Bayes matcher's corpus of terms, it cannot identify the category x belongs to

The subsequence matcher, however, could correctly identify x as belonging to the category "Soundcard" Sndblster is a subseqence of Soundblaster, which is a term that strongly indicates that the element represents "soundcard"

We have also seen in the previous section (5 3 2) that the neural network matcher's ability to generalise from its training data enables it to identify elements in situations where the Naive Bayes matcher or the subsequence matcher cannot

There are many such scenarios where one matcher can identify elements when one or both of the other matchers cannot

By selectıng the output from other matchers when the naıve bayes matcher cannot categorıse the ınput data, the composıte matcher can categorıze a wıder range of data than any of the ındıvıdual matchers

5.5 Evaluatıon of system

The Naıve Bayes algorıthm ıs hıghly effectıve at categorısıng textual data However, ıt was found durıng testıng that there were some cırcumstances under whıch ıt was ıneffectıve

Text contaıns dıfferent terms whıch are strongly ındıcatıve that the text belongs to multıple categorıes

Text contaıns no terms whıch appear ın the matcher's corpus

Under these cırcumstances the matcher would gıve ambıguous results In the first scenarıo, the matcher would assıgn a probabılıty of 1 0 to multıple categorıes For example, when the Naıve Bayes matcher trıes to classıfy "Server class machıne wıth 512MB RAM", ıt calculates that the probabılıty of thıs belongıng to the categorıes RAM and Harddrıve are both 1 0 In the second scenarıo, the Naıve Bayes Matcher assıgns a probabılıty of 0 to every category

The Neural Net matcher can generalıse It can "fill-ın" holes ın the traınıng dataset Thıs means that even ıf ıt ıs tryıng to categorıse data that does not appear ın ıts traınıng data, ıt can make ınferences from the dataset and return a predıctıon based on thıs It also does not return probabılıtıes of 1 0 for multıple categorıes when the data ıt ıs tryıng to categorıse contaıns terms ındıcatıng that ıt belongs to multıple categorıes The Neural Net matcher can successfully categorıse data under both of the cırcumstances that the Naıve Bayes cannot

The substrıng matcher ıs the most effectıve of the three matchers at categorısıng unknown elements wıth lengthy textual values

Each of the matchers can correctly categorise data which one or both of the other matchers cannot This is illustrated in the results obtained when the application was tested (see Appendix B) When the results returned by the individual matchers are not unanimous, the composite matcher picks the result that is statistically most likely to be correct Results obtained during testing clearly indicate that the accuracy rate of the composite matcher is higher than any of the individual matchers Using the composite matcher to select the individual matcher application which is most likely to be correct enables the application to categorise a wider range of data than a single matching algorithm

6. Results

Usıng Google, we searched for random web pages whıch descnbed hardware requırements for software applıcatıons The schema matcher was tested wıth data from these pages representıng the followıng devıce attnbutes

- CPU
- Harddnve space
- RAM
- Operatıng System
- Colour Depth
- Resolutıon
- Sound
- Vıdeo Card
- Network Connectıon

The test cases were text stnngs descnbıng the devıce attnbutes above, and were grouped accordıng to dıfficulty The more words ın the ınput, the more dıfficult ıt ıs to classıfy For example, the matcher categonses "128MB" wıth a greater degree of certaınty than "It ıs recommended that you have at least 128 MB for Wındows 2000', because there ıs less "noıse" ın the form of extra words

CATEGORY 1 (EASIEST)

Data contaıns words strongly ındıcatıng that data belongs to a partıcular category, and lıttle else Eg "128 MB", "300 Mhz", "Soundblaster"

CATEGORY 2

Data contaıns words strongly ındıcatıng that data belongs to a partıcular category, and 1-5 other words Eg "300 Mhz or greater", "1 5 GB Dısk space for setup", "100MB of free space or greater"

CATEGORY 3

Data contaıns words strongly ındıcatıng that element belongs to a partıcular category, and 6-10 other words Eg "150 MB free hard dısk space plus space for saved games", "Your system should have at least 32 Mb RAM"

CATEGORY 4 (HARDEST)

Instance level data contaıns words whıch strongly ındıcate that element belongs to multıple categorıes For example, "233 MHz or faster for Wındows 2000", "Sound card supported under Wındows NT"

For testıng, the Clıent Servıce Capabılıty Matcher had to classıfy 181 unknown elements based on theır values Thıs ıs a breakdown of how many of these elements belonged to each category

CATEGORY 1	33
CATEGORY 2	85
CATEGORY 3	48
CATEGORY 4	15

Appendıx B of thıs thesıs contaıns the data used for testıng, and what each of the semı-automatıc matchers and the composıte matcher classıfied these unknown elements as

The test data was selected from a wıde range of heterogeneous sources, so that the tests would be a reflectıon of the systems effectıveness ın a real-world scenarıo, where ıt would be medıatıng between many dıfferent data sources It also contaıns test cases that provıde a range of dıfferent challenges for the matchers, ın order to demonstrate the applıcatıon's effectıveness wıth dıfferent data posıng a varıety of challenges Some examples of the kınds of test data used are

Abbreviations of terms in the rules base
Misspellings of terms in the rules base
Terms which are not present in the rules base
Data containing terms indicating that the data belongs to multiple categories
Vague data, with no terms strongly indicating that the data belongs to a particular category
Long data, which has a greater probability of being incorrectly classified (the terms which are strong indicators of the category the data belongs to are buried in "noise")

The procedure for compiling the initial rules base was as follows

Determine the attributes that represent the properties of devices and web content (eg screen resolution, network speed, sound support) These represent the "canonical form" of element names in device/web content profiles

For each "canonical form" of a device/web content attribute, create rules in the rules base matching the canonical form of an attribute to a synonym (RAM=Memory, HardDisk=HardDrive)

Table 6 1 shows the performance of the matchers on each category of test data

When creating the training data for the individual matchers, we sought to maximise the accuracy of the composite matcher, not the accuracy of the individual matchers The figures for these matchers do not indicate the optimum performance of these algorithms

Table 6 1 Test results for matchers

Test Data	Neural Net	Naıve Bayes	Subsequence Matcher	Composıte Matcher
Category 1	30/33 (91%)	29/33 (88%)	24/33 (73%)	30/33 (91%)
Category 2	59/85 (69%)	69/85 (81%)	72/85 (85%)	80/85 (94%)
Category 3	34/48 (71%)	28/48 (58%)	40/48 (83%)	43/48 (90%)
Category 4	14/15 (93%)	9/15 (60%)	6/15 (30%)	13/15 (87%)

We observe that the the composıte matcher maıntaıns a consıstent level of accuracy for all lengths of ınput data The performance of the ındıvıdual matchers however, fluctuates wıth the length of the ınput For categorıes 1,2 and 3 of test data, the composıte matcher ıs at least as accurate (and ın most cases more accurate) than any of the ındıvıdual matchers, ıllustratıng the benefits of combınıng the results of the ındıvıdual matchers The composıte matcher performed slıghtly worse than the neural net matcher ın test cases belongıng to category 4 The composıte matcher had an accuracy rate of 87% compared to the neural net matcher's accuracy rate of 93% When we consıder that the Naıve Bayes matcher's accuracy rate ın thıs category was 60% and the subsequence matcher's accuracy rate was 30% however, the effectıveness of the composıte matcher's algorıthm for decıdıng whıch of the three matcher's ıs most lıkely to be correct ıs clear

Category 4 ıs dıfferent from the first three categorıes, ıt ıs a set of test cases that are dıfficult not because of theır length, but because they contaın terms that strongly ındıcate the data belongs to multıple categorıes (eg "128MB RAM for Wındows 2000") The neural network matcher performs the strongest here, ıllustratıng that ıts abılıty to generalıse allows ıt to make matches ın sıtuatıons where the naıve bayes matcher and the substrıng matcher faıl

New rules were successfully added to the rules base based on the output from the semı-automatıc matchers If an element from a devıce/web content profile had a name that dıd not appear ın the Clıent Servıce Capabılıty Matcher's ontology, and a mappıng between this element and an element in the Client Service Capability Matcher's ontology was suggested by the Composite Matcher to the user three times, the Rule Generator created a new rule indicating that these two elements were semantically equivalent The performance of the Client Service Capability Matcher when processing profiles from sources using this previously unknown element was improved because the Client Service Capability Matcher could now automatically match this element to a term in its own ontology without invoking the semi-automatic matchers The rules based matcher is quicker than the semi-automatic matchers, and does not require a human user to accept or reject its matches

Considering that neither the number of ontologies the Client Service Capability Matcher mediates between nor the terms contained in these ontologies is specified, we cannot guarantee that the rules based matcher will ever be able to automatically match all elements between all ontologies (it is doubtful that even a human could) However, through the addition of new rules to the rules based matcher the Client Service Capability Matcher was able to match terms in ontologies that it could not beforehand, illustrating the ability of the system to improve its performance by updating the rules base based on user interaction with the semi-automatic matchers

7. Conclusions and future work

There are many ontologies representing device and web content properties in existence, for example the ontologies used by UPnP[29], FIPA Device Ontology Specification[30], OWL-S [31] There are also a variety of technologies whose purpose is to describe devices and web content (eg UDDI [36], CC/PP [37], Jini [38], Service Location Protocol [39]) We manually examined the ontologies used by these various specifications These ontologies are not standardised, they may use different terms to represent the same concepts/objects One ontology might represent a device's memory by using the term 'RAM', where another may use the term 'Memory' Some specifications do not prescribe the use of a particular vocabulary, and to some extent allow the users to use whatever vocabulary they wish (eg CC/PP, Jini)

It is difficult to construct applications which recognise the meaning of terms from all these sources without explicit rules declaring semantically equivalent terms

The W3C's Device Independence activity [1] recommends that web-content be tailored according to the audio-visual capabilities and the input/output modalities available to each device This personalisation of web content must be carried out by applications which access metadata describing both the device and the web-content it is accessing to determine what changes (if any) to the web-content are required to enable the user of the device to access the web-content In the absence of standardised ontologies for describing devices and web-content, personalisation applications will frequently encounter meta-data that they cannot understand, and they will be unable to determine what (if any) transformations to the web-content are necessary

In this thesis we have described an architecture which can process device and web service RDF profiles using non-standardised ontologies, and identify semantic mappings between element names used in these profiles, and its own ontology The system checks to see if an element name is present in its ontology If it is not, it consults a combination of semi-automatic matchers (a naive bayes matcher, a neural net matcher, and a subsequence matcher) in order to determine what term in its ontology the unknown element name is most lıkely to be semantıcally equıvalent to New semantıc mappıngs can be added to the ontology based on ınteractıon wıth the end-user

The followıng components form the Clıent Servıce Capabılıty Matcher

- An RDF parser, whıch takes as ınput an RDF document and extracts the names of elements and the values assocıated wıth them RDF was chosen as the language for descrıbıng devıces and servıces because ıt ıs suffıcıently expressıve not only to model exıstıng devıces and servıces but also those whıch may appear ın future

- A rules based matcher, whıch ıs a lookup table ındıcatıng semantıc mappıngs between terms Thıs ıs an ontology of terms ın the domaın of devıce capabılıtıes and requırements

If an element name ın an RDF profile ıs not present ın the table used by the rules based matcher, three semı-automatıc matchers are used to suggest possıble mappıngs between the unknown term and terms ın the system's ontology, based on the value assocıated wıth the unknown term These semı-automatıc matchers are

- A Naıve Bayes matcher, whıch uses a probabılıstıc algorıthm based on Bayes Theorem, whıch has been wıdely used ın text classıficatıon problems

- A Subsequence matcher, based on the Longest Common Subsequence (LCS) algorıthm The LCS ıs computed between an unknown term and every term ın the system's ontology The LCS ıs used as part of a calculatıon whıch takes ınto account the length of the unknown term ın order to generate a sımılarıty score between the two terms beıng compared The term ın the ontology whıch has the hıghest sımılarıty score wıth the unknown term ıs chosen by the subsequence matcher as beıng the most lıkely to be semantıcally equıvalent to the unknown term

- A Neural Net matcher, ımplementıng the neural net algorıthm to classıfy data The Neural Net matcher converts textual data ınto numerıc data ın order to classıfy ıt

- The composite matcher decides which of the results returned by the three semi-automatic matchers is most likely to be correct, and suggests this to the end user as a possible match for the unknown term

- The rule generator creates new mappings in the rules base based on user interaction with the semi-automatic matchers The system's ontology will grow to include previously undefined terms This is a feature which is not present in the schema matching tools in existence at present

- The compatibility gauge evaluates the device's compatibility with the service based on the attributes retrieved from the RDF documents describing them by the rules based and semi-automatic matchers

Applications such as the Client Service Capability Matcher enable personalisation applications to process metadata from a wide range of heterogeneous sources, and can aid machines to perform the labour-intensive task of customising web-content for individual devices

For average test cases (test categories 1-3), our schema matching application was successfully able to categorise 92% of test cases, while it had an accuracy of 87% in cases where the input contained terms indicating the data belonged to multiple categories

The composite matcher's level of performance was consistent when deployed against data from heterogeneous sources The test data was obtained from random web-pages describing device specifications and requirements for applications The composite matcher's performance also remained consistent when it was tested with simple test cases (eg '128MB') and when it was tested with more difficult test cases where the input was longer and/or contained terms which indicated the input belonged to multiple categories (eg '64 Megabytes on Windows 95 or 98, 128 Megabytes on Windows NT') The composite matcher was also more accurate than any of the individual matchers in most cases, and was quite effective in determining which of the results provided by the three semi-automatic matcher's was most likely to be correct The results obtained during testing (outlined in chapter 6) clearly show the effectiveness of our application

7.1 Suggested improvements

Some terms in a device/service profile may map to more than one term in our ontology For example, the term '1024x768 resolution with 16-bit colour' maps to two terms in our ontology, resolution ('1024x768 resolution') and colour ('16-bit colour') At present only 1-1 matches can be made between elements in profiles and the Client Service Capability Matcher's ontology Implementing n-1 mappings is an area requiring further research in this area

The following steps could also be taken to increase the accuracy of the neural net matcher

- Create a larger training set At present, the training set consists of 988 examplars For a neural net with 59 inputs, a larger training set may improve the accuracy of the neural net

- Compare the input data to a different set of terms Perhaps if we generate similarity scores for a larger, or more varied, set of terms, the accuracy and generalisation ability of the neural net will be improved

- Different net architecture We have not experimented with different net architectures Using neural nets with different numbers of hidden layer nodes, or more than one hidden layer, may improve the matchers performance

We also plan to investigate if enlarging the corpus of terms for the Naive Bayes and Subsequence matchers will improve their accuracy

The data that the matcher uses to train the neural net and to perform matches using the Naive Bayes and Subsequence matchers must be updated periodically The semi-automatic matchers suggest matches based only on this data This data must be representative of the unknown terms likely to be encountered The data should be constantly updated to reflect this Gathering all the necessary data can be a time-consuming process For the purposes of testing this system, data was obtained from web pages Ensuring that the data is properly prepared for the neural net matcher is also time-consuming The input file representing the neural net's training data must be carefully prepared, the order that data appears in this file greatly affects the neural net's performance, so care must be taken when preparing this file Further research is required into how this training data can be updated automatically

The composite matcher always suggests a match for unknown elements in a profile, even if the unknown element does not belong to any of the categories the matcher identifies A further improvement to the application would be to enable it to recognise elements which do not belong to any of the categories it recognises, and to not generate semantic mappings for these unknown elements

The Semantic Web is an area in which semi-automatic schema matching techniques are of value Web-content on the Semantic Web is annotated with metadata that allows machines to "understand" the semantic meaning of that data, and to reason about and process data from a wide range of heterogeneous sources in ways that machines currently cannot The widespread annotation of metadata to web-content will result in a proliferation of ontologies There are a number of research efforts aiming to produce means of expressing these ontologies (eg OWL, DAML-S, DARPA Agent Markup)

It is desirable that the performance of schema-matching applications improves over time Our application can add new rules to its ontology based on users interaction with the semi-automatic matcher However, this feature can be abused by those who wish to manipulate the behaviour of the matcher (for example, by creating false mappings) More work is needed to devise mechanisms that allow the performance of the system to be improved by the interaction with end users, while not permitting end users to create fake mappings

The generation of N 1 mappings semi-automatically is also a direction that further research in the field of schema matching will follow

Using schema matching techniques to perform matches between large numbers of data sources is also an avenue that deserves further investigation At present, schema matching systems are used to mediate between relatively small numbers of data sources Deploying schema matching applications on the internet will require them to be scalable in terms of both the number of data sources that semantic mappings are being made between and the size of those sources

8. References:

[1] W3C Devıce Independence Workıng Group, Devıce Independence Prıncıples W3C Workıng Group Note 1 September 2003 http //www w3 org/TR/2003/NOTE-dı-prınc-20030901/ (accessed 5 December 2003)

[2] M Frank, P Szekely, and R Neches a Baoshı Yah a Juan Lopez *Web- scrıpter World-wıde grass-roots ontology translatıon vıa ımplıcıt enduser alıgnment* In M Frank, N Noy, and S Staab, edıtors, Proceedıngs of the Semantıc Web Workshop 2002

[3] M Dean, G Schreıber, S Bechhofer, F v Harmelen, J Hendler, I Horrocks, D McGuınness, P Patel-Schneıder, L Steın Web Ontology Language (OWL) Reference Versıon W3C Candıdate Recommendatıon 18 August 2003 http //www w3 org/TR/owl-ref/ (accessed 5 December 2003)

[4] E Mıller, Semantıc Web Actıvıty Statement 9 November 2003 http //www w3 org/2001/sw/Actıvıty Accessed 5 December 2003

[5] W3C RDF Model and Syntax Workıng Group, Resource Descrıptıon Framework (RDF) Model and Syntax Specıficatıon, W3C Recommendatıon 22 February 1999 http //www w3 org/TR/REC-rdf-syntax/ (accessed 5 December 2003)

[6] P Mıtra, G Wıederhold, and J Jannınk *Semı-automatıc ıntegratıon of knowledge sources* In Proc of the 2nd Int Conf On Informatıon FUSION'99, 1999

[7] Tova Mılo and Sagıt Zohar *Usıng schema matchıng to sımplıfy heterogeneous data translatıon* In Proc of the Int Conf on Very Large Data Bases (VLDB), New York Cıty, USA, 1998

[8] A Doan, P Domıngos, and A Levy *Reconcılıng Schemas of Dısparate Data Sources A Machıng-Learnıng Approach* In SIGMOD, pages 509-520, 2001

[9] Lı W, Clıfton C *SemInt A Tool for Identıfyıng Attrıbute Correspondences ın Heterogeneous Databases Usıng Neural Network* Data and Knowledge Engıneerıng 33 1, 49-84, 2000

[10] D Hırschberg Serıal Computatıons of Levenshteın Dıstance In Pattern Matchıng Algorıthms Oxford Unıversıty Press, 1997

[11] J Madhavan, P Bernsteın, and E Rahm *Generıc Schema Matchıng wıth Cupıd* In Proceedıngs of the Internatıonal Conference on Very Large Databases (VLDB), 2001

[12] D D Lewıs *Naıve Bayes at forty The ındependence assumptıon ın ınformatıon retrıeval* In ECML-98 Proceedıngs of the Tenth European Conference on Machıne Learnıng, pages 4--15, Chemnıtz, Germany, Aprıl 1998 Sprınger

[13] Peng, F and Schuurmans, D , (2003) *Combınıng Naıve Bayes and n-Gram Language Models for Text Classfıcatıon* submıtted to The 25th European Conference on Informatıon Retrıeval Research (ECIR)

[14] A McCallum and K Nıgam 1998 A comparıson of event models for naıve Bayes text classıficatıon In *AAAL98 Workshop on Learnıng]or Text Categorızatıon* Tech rep WS-98-05, AAAI Press http //www cs cmu edu/Nmccallum

[15] W W Cohen and H Hırsh Joıns that generalıze Text classıficatıon usıng whırl In Proc of the FourthInt Conf on Knowledge Dıscovery and Data Mınıng (KDD-98), 1998

[16] P Domıngos and M Pazzanı Beyond ındependence Condıtıons for the optımalıty of the sımple Bayesıan classıfier In Proceedıngs of the Thırteenth Internatıonal Conference on Machıne Learnıng, pages 105-112, Barı, Italy, 1996 Morgan Kaufmann

[17] Teuvo Kohonen Adaptıve, assocıatıve, and self-organızıng functıons ın neural computıng Applıed Optıcs, 26 4910-4918, 1987

[18] M Sahamı, S Dumaıs, D Heckerman, E Horvıtz A Bayesıan Approach to Fılterıng E-Maıl Learnıng for Text Categorızatıon – Papers from the AAAI Workshop, pages 55-62, Madıson Wısconsın AAAI Technıcal Report WS-98-05

[19] G Escudero, L M arquez, and G Rıgau Naıve Bayes and Exemplar-Based Approaches to Word Sense Dısambıguatıon Revısıted Proceedıngs of the 14th European Conference on Artıficıal Intellıgence, ECAI, Berlın, Germany, 2000

[20] R Chınnıcı, M Gudgın, J Moreau, J Schlımmer, S Weerawarana, Web Servıces Descrıptıon Language (WSDL) Versıon 2 0 Part 1 Core Language, W3C Workıng Draft 10 November 2003 http //www w3 org/TR/wsdl20/ (accessed 9 February 2004)

[21] T Bray, J Paolı, C M Sperberg-McQueen, E Maler, F Yergeau, Extensıble Markup Language (XML) 1 0 (Thırd Edıtıon), W3C Recommendatıon 4 February 2004 http //www w3 org/TR/2004/REC-xml-20040204/ (accessed 9 February 2004)

[22] N Mıtra, SOAP Versıon 1 2 Part 0 Prımer, W3C Recommendatıon 24 June 2003 http //www w3 org/TR/soap12-part0/ (accessed 9 February 2004)

[23] R Chınnıcı, Java(TM) API for XML-Based RPC Specıficatıon 1 1 Fınal Draft, 21 Aprıl 2003 http //java sun com/xml/downloads/jaxrpc html#jaxrpcspec09 (accessed 9 February 2004)

[24] E Armstrong, Java API for XML Processing
http //java sun com/webservices/docs/1 0/tutorial/doc/JAXPIntro html (accessed 9 February 2004)

[25] ANSI X 3 135-1992, American National Standard for Information Systems – Database Language SQL, American National Standards Institute, New York, 1992

[26] JDBC API, Sun Microsystems,http //www javasoft com/products/jdbc/ (accessed 13 February 2004)

[27] Jude W Sharvik, Thomas G Dietterich Readings in Machine Learning Morgan Kaufmann, 1990

[28] Java Object Oriented Neural Engine API, http //www jooneworld com/ (accessed 13 February 2004)

[29] UPnP Forum, UPnP Basic Device Definition v1 0, 12 December 2002
http //www upnp org/standardizeddcps/documents/BasicDevice-1 0 pdf (accessed 16 June 2004)

[30] Foundation for Intelligent Physical Agents, FIPA Device Ontology Specification, 10 May 2002 http //www fipa org/specs/fipa00091/xc00091c pdf (accessed 16 June 2004)

[31] Web Ontology Working Group, W3C, OWL-S Semantic Markup for Web Services, November 2003, www daml org/services/owl-s/1 0/owl-s pdf (accessed 16 June 2004)

[32] J Clark, XSL Transformations (XSLT) Version 1 0, 16 November 1999,
http //www w3 org,TR/xslt (accessed 18 September 2004)

[33] R Lewıs, Authorıng Challenges for Devıce Independence, W3C Workıng Group Note 1 September 2003, http //www w3 org/TR/2003/NOTE-acdı-20030901/ (accessed 18 September 2004)

[34] R Hanrahan, R Merrıck Authorıng Technıques for Devıce Independence, W3C Workıng Group Note 18 February 2004, http //www w3 org/TR/2004/NOTE-dı-atdı-20040218/ (accessed 18 September 2004)

[35] T Bray, D Hollander, A Layman Namespaces ın XML, 14 January 1999, http //www w3 org/TR/REC-xml-names/ (accessed 18 September 2004)

[36] T Bellwood, L Clement, C Von Rıegen, UDDI Versıon 3 0 1, UDDI Spec Technıcal Commıttee Specıfıcatıon, 14 October 2004, http //uddı org/pubs/uddı-v3 0 1-20031014 htm (accessed 18 September 2004)

[37] G Klyne, F Reynolds, C Woodrow, H Ohto, J Hjelm, M H Butler, L Tran, Composıte Capabılıty/Preference Profiles (CC/PP) Structure and Vocabularıes, W3C Workıng Draft 25 March 2003, http //www w3 org/TR/2003/WD-CCPP-struct-vocab-20030325/ (accessed 18 September 2004)

[38] Jını Technology Core Platform Specıfıcatıon, http /wwws sun com/software/jını/specs/ (accessed 18 September 2004)

[39] E Guttman, C Perkıns, Servıce Locatıon Protocol Versıon 2, June 1999, http //www openslp org/doc/rfc/rfc2608 txt (accessed 18 September 2004)

[40] K M Tıng, I H Wıtten Issues ın stacked generalızatıon Journal of Artıfıcıal Intellıgence Research, 10 271-289, 1999

[41] M R Genesereth, R E Fıkes, Knowledge Interchange Format, Reference Manual, 1992

[42] A LeHors, I Jacobs D Raggett, HTML 4 01 Specıfıcatıon, W3C Recommendatıon 24 December 1999, http //www w3 org/TR/html4/, (accessed 18 September 2004)

[43] Java Web Servıces Development Pack, http //java sun com/webservıces/jwsdp/ındex jsp, (accessed 18 September 2004)

[44] Java Remote Method Invocatıon, http //java sun com/products/jdk/rmı/, (accessed 18 September 2004)

Appendix A : Sample WSDL Document

Here follows a sample WSDL document It describes a web service which returns the temperature in an area when given that area's zip-code

```
<?xml version='1 0' encoding='UTF-8' ?>
<!-- Generated 08/16/01 by Microsoft SOAP Toolkit WSDL File Generator, Version
1.02 813 0 -->
<definitions  name ='VB6Weather'
  targetNamespace = 'http //tempuri org/wsdl/'
 xmlns wsdlns='http //tempuri org/wsdl/'
 xmlns typens='http //tempuri org/type'
 xmlns soap='http //schemas xmlsoap org/wsdl/soap/'
 xmlns xsd='http //www w3 org/2001/XMLSchema'
 xmlns stk='http //schemas microsoft com/soap-toolkit/wsdl-extension'
 xmlns='http //schemas xmlsoap org/wsdl/'>

  <types>
   <schema targetNamespace='http://tempuri org/type'
    xmlns='http //www w3 org/2001/XMLSchema'
    xmlns.SOAP-ENC='http //schemas xmlsoap org/soap/encoding/'
    xmlns wsdl='http //schemas xmlsoap org/wsdl/'
    elementFormDefault='qualified'>
   </schema>
  </types>

  <message name='VB6Weather GetTemperature'>
   <part name='zipcode' type='xsd string'/>
   <part name='celsius' type='xsd boolean'/>
  </message>
```

```
<message name='VB6Weather GetTemperatureResponse'>
 <part name='Result' type='xsd float'/>
</message>

<portType name='VB6WeatherSoapPort'>
 <operation name='GetTemperature' parameterOrder='zipcode celsius'>
  <input message='wsdlns VB6Weather GetTemperature' />
  <output message='wsdlns VB6Weather GetTemperatureResponse' />
 </operation>

</portType>

<binding name='VB6WeatherSoapBinding' type='wsdlns·VB6WeatherSoapPort'
>
 <stk binding preferredEncoding='UTF-8'/>
 <soap binding
   style='rpc'
   transport='http //schemas xmlsoap.org/soap/http' />
 <operation name='GetTemperature' >
  <soap operation
  soapAction='http //tempuri org/action/VB6Weather.GetTemperature' />
  <input>
   <soap body use='encoded' namespace='http //tempuri org/message/'
  encodingStyle='http //schemas xmlsoap org/soap/encoding/' />
  </input>

  <output>
   <soap·body use='encoded' namespace='http //tempuri.org/message/'
  encodingStyle='http //schemas xmlsoap org/soap/encoding/' />
```

```
    </output>

   </operation>
  </binding>

  <service name='VB6Weather' >
 <port
     name='VB6WeatherSoapPort'
     binding='wsdlns VB6WeatherSoapBinding' >
    <soap address location='http //localhost/webtest/vb6weather/VB6Weather.ASP'
/>
   </port>
  </service>

</definitions>
```

Appendix B: Test Results

Category 1 Test Results

Test data	Category of test data	Neural Net Result	Naive Bayes Result	Substring Matcher result	Did the composite matcher pick the right result?
Pentium 300 MHz	CPU	CPU	CPU	CPU	YES
128 MB RAM	RAM	RAM	RAM	VideoRAM	YES
32-bit O/S - Windows XP, ME, 2000, 98, or NT 4 0	Operating System	Operating System	Operating System	Colour	YES
64MB RAM	RAM	RAM	RAM	RAM	YES
Pentium 600 MHz	CPU	CPU	CPU	CPU	YES
128 MB RAM	RAM	RAM	RAM	VideoRAM	YES
Windows 98	Operating System	Operating System	Operating System	Operating System	YES
32 MB RAM	RAM	RAM	RAM	VideoCard	YES
Microsoft Windows 95/98/NT/2000/XP/M E	Operating System	Operating System	Operating System	Harddrive	YES
8 MB RAM	RAM	RAM	Video Card	VideoRAM	NO
266 MHz Pentium	CPU	CPU	CPU	CPU	YES
Windows 98, ME, 2000, or XP	Operating System	Operating System	Operating System	Operating System	YES
50MB available	Harddrive	Harddrive	Harddrive	Harddrive	YES
16-bit (high colour)	Colour	Colour	Colour	Video Card	YES
800 x 600 resolution	Resolution	Resolution	Resolution	Resolution	YES
16-bit sound card	Sound	Sound	Sound	Sound	YES
56 6 kb/s modem or LAN connection	Network Connection	Sound	Network Connection	Network Connection	YES
Pentium 90	CPU	CPU	CPU	CPU	YES
16 MB of RAM	RAM	RAM	RAM	Video RAM	YES
14 4 Kbps	Network Connection	Network Connection	Network Connection	Network Connection	YES
16 MB	RAM	Video RAM	RAM	Video RAM	YES
64MB RAM	RAM	RAM	RAM	RAM	YES

Windows NT	Operating System	Operating System	Operating System	Operating System	YES
VGA (640x480)	Resolution	Resolution	Video Card	Resolution	NO
486DX/66 MHz	CPU	CPU	CPU	CPU	YES
MS Windows 98, NT or 2000	Operating System	Operating System	Operating System	Operating System	YES
64 MB of ram	RAM	RAM	CLASH	RAM	YES
Windows 95/98 or Windows NT/2000	Operating System	Operating System	Operating System	Operating System	YES
800 MHz CPU	CPU	CPU	CPU	CPU	YES
9 GB	Harddrive	Resolution	Harddrive	Harddrive	YES
Windows 95/98/NT operating system	Operating System	Operating System	Operating System	Operating System	YES
800 MHz processor	CPU	CPU	CPU	CPU	YES
45 MB	Harddrive	Harddrive	RAM	Harddrive	NO

Accuracy of matchers with category 1 test data

Neural Net	Naive Bayes	Subsequence Matcher	Composite Matcher
30/33 (91%)	29/33 (88%)	24/33 (73%)	30/33 (91%)

Category 2 Test Results

Test data	Category of test data	Neural Net Result	Naive Bayes Result	Substring Matcher result	Did the composite matcher pick the right result?
200 MB available disk space	Harddrive	Harddrive	Harddrive	Harddrive	YES
Windows(R) 95 or Windows 98 (Windows NT is	Operating System	Operating System	CLASH	Operating System	YES

not supported)					
Sound card with DirectSound(TM) support	Sound	Sound	CLASH	Sound	YES
300 MHz or faster Pentium processor	CPU	CPU	CPU	CPU	YES
800 x 600 or Higher	Resolution	Resolution	Video Card	Resolution	NO
Pentium 200 MHz or higher	CPU	CPU	CPU	CPU	YES
Windows 98, Windows 98SE*, Windows Me, Windows 2000, or Windows XP (Home and Pro)	Operating System	Operating System	Operating System	Operating System	YES
70MB free disk space	Harddrive	Harddrive	Harddrive	Harddrive	YES
Windows 2000 Professional	Operating System	Operating System	Operating System	Operating System	YES
56k v 90 modem or network Internet connection	Network Connection	Sound	Network Connection	Network Connection	YES
300MHz Pentium II® or faster	CPU	CPU	CPU	CPU	YES
30MB free disk space	Harddrive	Harddrive	Harddrive	Harddrive	YES

64 MB RAM (minimum)	RAM	RAM	Video Card	RAM	NO
Microsoft Windows® NT/2000 (preferred) or Windows 9x	Operating System	Operating System	Operating System	Operating System	YES
10 MB of hard disk space	Harddrive	Harddrive	Harddrive	Harddrive	YES
56kbps Modem or higher	Network Connection	RAM	Network Connection	Network Connection	YES
150 MB Free Hard Drive Space	Harddrive	Harddrive	Harddrive	Harddrive	YES
32 MB Accelerated Video Card w/ Open GL	Video Card	Video RAM	Video Card	Video Card	YES
Pentium based processor or better (300Mhz)	CPU	CPU	CLASH	CPU	YES
150 Free Hard Drive Space	Harddrive	Video Card	Harddrive	Harddrive	YES
Server class machine with 512MB RAM	RAM	RAM	CLASH	Operating System	NO
Pentium III, 450 MHz or higher processor	CPU	CPU	CPU	CPU	YES
Pentium III 800	CPU	CPU	CPU	CPU	YES

MHz Dual Processor					
Pentium-class, minimum 133 MHz (megahertz) or faster	CPU	Harddrive	CPU	CPU	YES
28 8/33 6 (kilobits per second) minimum	Network Connection	CPU	Network Connection	Network Connection	YES
8 Mb of memory	RAM	VideoRAM	RAM	RAM	YES
300 mhz or greater	CPU	CPU	CPU	CPU	YES
500 MB Hard Drive	Harddrive	Harddrive	Harddrive	Harddrive	YES
Windows 95 or higher	Operating System	CPU	Operating System	Operating System	YES
20 MB of disk space	Harddrive	Harddrive	Harddrive	Harddrive	YES
Requires 10 MB of disk space	Harddrive	Network Connection	Harddrive	Harddrive	YES
Pentium 233MHz processor or higher	CPU	CPU	CPU	CPU	YES
64MB RAM or more	RAM	RAM	RAM	RAM	YES
Pentium 400 MHz or more	CPU	CPU	CPU	CPU	YES
15 MB free hard	Harddrive	Harddrive	Harddrive	Harddrive	YES

disk space					
Intel Pentium-II 350 system or faster	CPU	CPU	CLASH	CPU	YES
Disk space for setup 1 5GB	Harddrive	Colour	Harddrive	Harddrive	YES
6 MB of disk space	Harddrive	Harddrive	Harddrive	Harddrive	YES
SoundBlaster (or equivalent) sound card	Sound	Sound	Sound	Sound	YES
Pentium-II 300Mhz	CPU	CPU	CPU	CPU	YES
Windows 95 or higher or NT4	Operating System	CPU	Operating System	Operating System	YES
128 MB or higher	RAM	RAM	RAM	Video RAM	YES
28 8 kbps modem or faster	Network Connection	Sound	Network Connection	Network Connection	YES
300 Pentium II or higher	CPU	Resolution	CPU	CPU	YES
1 Gig or larger	Harddrive	Colour	Harddrive	Harddrive	YES
Resolution 800x600 or higher	Resolution	Resolution	CLASH	Resolution	YES
15 Megabytes of hard drive space	Harddrive	Harddrive	Harddrive	Harddrive	YES
A 266Mhz Pentium II processor or	CPU	CPU	CLASH	CPU	YES

better					
A 16MB OpenGL compatible Graphics card	Video Card	Video Card	Video Card	Video Card	YES
800x600 or larger color display	Resolution	Colour	CLASH	Resolution	NO
16 megs RAM or more	RAM	RAM	CLASH	VideoRAM	YES
100MB of free space minimum	Harddrive	Harddrive	Harddrive	Network Connection	YES
20 GB Hard Drive	Harddrive	Video Card	Harddrive	Harddrive	YES
2 GB or greater	Harddrive	Resolution	Harddrive	Harddrive	YES
Intel Pentium II 233mhz or equivalent	CPU	CPU	CLASH	CPU	YES
Intel Pentium II/Celeron 300mhz or higher	CPU	CPU	CPU	CPU	YES
50MB for software and index	Harddrive	Harddrive	Harddrive	Harddrive	YES
56 6k Dial-up connection or faster	Network Connection	Resolution	Network Connection	Network Connection	YES
32 MB or more	RAM	RAM	RAM	Video RAM	YES
800 x 600 or higher	Resolution	Resolution	Resolution	Resolution	YES

Pentium Celeron 500 MHz	CPU	CPU	CPU	CPU	YES
At least a connection at 57 6 kbps	Network Connection	Network Connection	Network Connection	Network Connection	YES
200 Megahertz Pentium-class or better	CPU	RAM	CLASH	CPU	YES
Minimum for text-mode 32MB	RAM	RAM	Harddrive	CPU	NO
VGA graphics (640 by 480 dots)	Video Card	Video Card	CLASH	Video Card	YES
1 GB available disk space	Harddrive	CPU	Harddrive	Harddrive	YES
display monitor capable of 1024x768 resolution	Resolution	Video Card	Resolution	Resolution	YES
80MB of available hard disk space	Harddrive	Harddrive	Harddrive	Harddrive	YES
Sound Blaster® or compatible sound card (16 or 32 bit)	Sound	Sound	CLASH	Sound	YES
SVGA Monitor (resolution of 800x600 or higher)	Resolution	Resolution	Resolution	Video Card	YES

Windows 95 or NT 4 0, Windows NT suggested	Operating System	Operating System	Operating System	Operating System	YES
Minimum 16 meg RAM	RAM	Resolution	RAM	CPU	YES
Minimum 8 meg RAM	RAM	Harddrive	RAM	CPU	YES
Audio capabilities (ie sound card and speakers)	Sound	Sound	Sound	Sound	YES
Monitor that displays at least 256 colors	Colour	Video Card	Colour	Colour	YES
20 Megabytes of free hard disk space	Harddrive	Harddrive	Harddrive	Video RAM	YES
Microsoft Windows 95 or greater	Operating System	Operating System	Operating System	Harddrive	YES
20 Gigabyte Hard drive (Free Space)	Harddrive	Video Card	Harddrive	Harddrive	YES
Windows 2000 Professional, Service Pack 4 or later	Operating System	Operating System	Operating System	Operating System	YES
1 GB or larger hard drive	Harddrive	Video Card	Harddrive	RAM	YES

133 MHz or higher Pentium-compatible CPU	CPU	CPU	CPU	Video RAM	YES
90 MHz or faster Pentium-Based PC	CPU	CPU	CPU	CPU	YES
30MB free hard disk space	Harddrive	Harddrive	Harddrive	Harddrive	YES
50 MB Available Hard Disk Space	Harddrive	Harddrive	Harddrive	Harddrive	YES
Pentium IV 1 7 GHz based system	CPU	CPU	CPU	CPU	YES

Accuracy of matchers with category 2 test data

Test Data	Neural Net	Naive Bayes	Subsequence Matcher	Composite Matcher
Category 2	59/85 (69%)	69/85 (81%)	72/85 (85%)	80/85 (94%)

Category 3 Test Results

Test data	Category of test data	Neural Net Result	Naive Bayes Result	Substring Matcher result	Did the composite matcher pick the right result?
150 MB free hard disk space plus space for saved games	Harddrive	Harddrive	Harddrive	Harddrive	YES

Hardware accelerated D3D compatıble 4MB vıdeo card wıth DırectDraw(TM) compatıble drıver	Vıdeo Card	Vıdeo Card	Vıdeo Card	Harddnve	YES
Pentıum 4, 1 3ghz (a hıgher-speed CPU normally results ın ımproved performance)	CPU	CPU	CPU	CPU	YES
256 MB RAM or hıgher (more memory normally results ın ımproved performance)	RAM	RAM	CLASH	RAM	YES
Wındows 2000 (Servıce Pack 3 or greater) and Wındows XP (Home or Professıonal)	Operatıng System	Operatıng System	Operatıng System	Operatıng System	YES
Up to 15 Mb of dısk space avaılable (dependıng on	Harddnve	Harddnve	Harddnve	Harddnve	YES

the use of CyberNOT)					
Approximately 10 megabytes of disk space is required for installi	Harddrive	Network Connection	CLASH	Harddrive	NO
A hard disk with at least 20 MB available for program installatio	Harddrive	Harddrive	CLASH	Harddrive	YES
16 MB TNT2-class OpenGL 1 2 compliant video card	Video Card	Video Card	CLASH	Video RAM	YES
An OpenGL accelerated video card (minimum 8MB VC RAM 16 MB recom	Video Card	Video Card	Video Card	Video Card	YES
128 megabytes (MB) of RAM for the operating system and services	RAM	RAM	CLASH	RAM	YES
Server with Pentium II 400	CPU	CPU	CLASH	CPU	YES

megahertz (MHz) or higher					
815 MB of available hard disk space for typical installation of a	Harddrive	Harddrive	Harddrive	Harddrive	YES
SVGA video card with 8MB video memory and video overlay capabilit	Video Card	Video Card	Video Card	Video Card	YES
50MB free hard disk space for a minimum installation	Harddrive	Harddrive	Harddrive	Harddrive	YES
At least 175 MB of free disk space to store the databases	Harddrive	Harddrive	CLASH	Harddrive	YES
486DX with a processing speed of 66 MHz or greater	CPU	CPU	CLASH	CPU	YES
128 MB of RAM with virtual memory on	RAM	RAM	RAM	VideoRAM	YES
120 MB or more	Harddrive	Harddrive	CLASH	Harddrive	YES

of available hard-disk space for installation.					
Minimum screen resolution of 800 x 600 pixels	Resolution	Resolution	Resolution	RAM	YES
5 MB of free Hard Disk space to install the program	Harddrive	Harddrive	Harddrive	Harddrive	YES
10 gigabytes (GB) free space on hard disk or higher	Harddrive	Sound	Harddrive	Harddrive	YES
10 GB (Gigabytes) of hard drive space is adequate	Harddrive	Video Card	CLASH	Harddrive	NO
Hard drive with minimum 2GB available for application and database	Harddrive	Video Card	Harddrive	Harddrive	YES
25 MB free (for Helper Application installation, if necessary)	Harddrive	Harddrive	CLASH	Harddrive	YES
100 MB free for	Harddrive	Harddrive	Harddrive	Network	YES

data files and temporary files				Connection	
Super VGA capable of providing 800 x 600 resolution	Video Card	Video Card	CLASH	Video Card	YES
486 DX2 66-MHz (or equivalent) processor	CPU	CPU	CPU	CPU	YES
Your system should have at least 32 Mb RAM	RAM	CPU	CLASH	RAM	NO
400 MHz Pentium II or better recommended	CPU	CPU	CLASH	CPU	YES
1 GB or more of available storage	Harddrive	Colour	CLASH	Harddrive	YES
64Mb minimum memory (more is better)	RAM	RAM	RAM	RAM	YES
Hard disk subsystem (400Mb or more),	Harddrive	RAM	Harddrive	Harddrive	YES
60 MB free disk space on your	Harddrive	Harddrive	Harddrive	Harddrive	YES

hard drive					
one 20GB IDE or SCSI hard drive or greater	Harddrive	Video Card	Harddrive	Harddrive	YES
1 GB minimum disk space per hard drive	Harddrive	Video Card	CLASH	Harddrive	NO
9 Gig SCSI Hard Drive space	Harddrive	Video Card	Harddrive	Harddrive	YES
128+ Megs of memory (Novell NetWare 4 xx/5 xx)	RAM	RAM	RAM	RAM	YES
100MB for software index and additional space for future index size	Harddrive	Harddrive	Harddrive	Network Connection	YES
32 Mbytes of RAM memory, additional memory is recommended	RAM	Video RAM	RAM	RAM	YES
Any computer running Windows 95 or later can run Musaios	Operating System	Operating System	RAM	Harddrive	NO
The program as	Harddrive	Video Card	Harddrive	Harddrive	YES

ınstalled takes 2 MB or less on your hard drıve					
500 MB hard dısk space for Multı-Market data	Harddrıve	Harddrıve	Harddrıve	Harddrıve	YES
Sıngle processor (Pentıum III, Pentıum 4/Xeon, AMD XP/MP) 400 MHz mınımum	CPU	CPU	CLASH	CPU	YES
Modem wıth a connectıon speed of 28 8k or hıgher	Network Connectıon	CPU	CLASH	Network Connectıon	YES
WınCross needs approxımately 50MB of dısk space	Harddrıve	Harddrıve	Harddrıve	Operatıng System	YES
Any 56K hardware modem - Stay away from Wın Modems!	Network Connectıon	Sound	CLASH	Network Connectıon	YES
200 MHz or faster Intel(R) Pentıum(R)	CPU	CPU	CPU	CPU	YES

MMX, Cyrix M2, or AMD(R) processor					

Accuracy of matchers with category 3 test data

Test Data	Neural Net	Naïve Bayes	Subsequence Matcher	Composite Matcher
Category 3	34/48 (71%)	28/48 (58%)	40/48 (83%)	43/48 (90%)

Category 4 Test Results

Test data	Category of test data	Neural Net Result	Naïve Bayes Result	Substring Matcher result	Did the composite matcher pick the right result?
233 MHz recommended for XP machines	CPU	CPU	CLASH	RAM	YES
Intel« Pentium 200-MHz or faster processor for audio	CPU	CPU	CPU	CPU	YES
233 MHz or faster for Windows 2000	CPU	CPU	CPU	CPU	YES
5 MB for Microsoft« Installer	Harddrive	Harddrive	Harddrive	RAM	YES
64 MB for NT	RAM	RAM	CLASH	RAM	YES
128MB (for Windows NT)	RAM	RAM	RAM	VideoRAM	YES
128 MB RAM required for Windows 2000 and XP	RAM	RAM	CLASH	VideoRAM	YES
486 (DX) CPU or higher for Windows ME	CPU	CPU	CLASH	VideoRAM	NO

Sound card supported under Windows NT	Sound	Operating System	Sound	Sound	YES
Graphics card with 64 MB RAM	Video Card	Video Card	Video Card	Video Card	YES
64 Megabytes on WindowsO 95 or 98, 128 Megabytes on Windows NT or	RAM	RAM	RAM	Harddrive	YES
150 Megabytes including Windows swapfile	Harddrive	Harddrive	Harddrive	RAM	YES
32 MB required for Windows 98 Operating System	RAM	RAM	Operating System	Operating System	NO
8 Megs of Memory (DOS/Windows 3 xx)	RAM	RAM	RAM	RAM	YES
19 Megabytes on hard disk for the Windows version	Harddrive	Harddrive	CLASH	Video RAM	YES

Accuracy of matchers with category 4 test data

Test Data	Neural Net	Naive Bayes	Subsequence Matcher	Composite Matcher
Category 4	14/15 (93%)	9/15 (60%)	6/15 (30%)	13/15 (87%)