[3] Weibel, Stuart. 'The State of the Dublin Core', in *D-Lib Magazine,* Volume 5 Number 4, April 1999. Paper accessed via http://www.dlib.org/dlib/april99/04weibel.html on 16 April 1999.

[4] Public Record Office Victoria, Ernst and Young and CSIRO. *Victorian Electronic Records Strategy Final Report 1999,* Public Record Office Victoria, 1998

Acknowledgements

We acknowledge the **contribution** of other Research Team members and associates, particularly Kate **Cumming,** APA(I) scholarship holder, Dr Nigel Ward and Dr Linda Bird, DSTC, and Geoff **Acland-Bell** (for the expert rendering of the conceptual models). We also acknowledge the input of members of the Project Steering Committee and our Expert Reference Group of Australian and international reviewers.

1 The Project involved collaboration between Chief **Investigators** Sue McKemmish, Monash University, and Ann **Pederson,UNSW,** and their industry partners from the National Archives of Australia, State Records Authority of NSW, Queensland State Archives, the Records Management Association of Australia and the Australian Council of Archives.

2 The acronym SPIRT derives from the name of the Research Grant which funded the Project, Strategic Partnership with Industry - Research & Training (SPIRT) Support Grant, which provides for joint funding by the Australian Research Council and the Industry partners.

3 Recordkeeping metadata is defined broadly to include all standardised information that identifies, authenticates, describes, manages and makes accessible documents created in the context of social and business activity. Recordkeeping metadata so defined has traditionally been captured and managed in both recordkeeping systems and archival control systems.

4 Within the archival community the **ACA/ASA** Committee on Descriptive Standards has endorsed the SPIRT Recordkeeping Metadata Schema as a framework for the Committee's future work on the development of domain specific recordkeeping metadata and archival descriptive standards. The Chair of this Committee has recently approached Standards Australia with a proposal to develop the SPIRT Recordkeeping Metadata Schema into a Framework Australian Standard for Recordkeeping Metadata.

5 The term "Schema" (plural schemata) is used to mean the semantic and structural definition of the metadata used to describe recordkeeping entities. A schema describes the names of metadata elements, how they are structured, their meaning and so on. The metadata community also refers to metadata schemata as metadata sets or specifications.

6 The Resource Description Framework (RDF) was developed by the World-Wide Web Consortium (W3C) to provide the foundation for metadata interoperability across different resource description communities, see: http://www.w3.org/RDF.

7 Object Role Modelling (ORM) takes a conceptual modelling approach that views the world in terms of objects and the roles they play. It is very expressive, enabling a high level of detail and rigorous analysis, and can be populated with data instances which thus allows for grounded validation.

8 For background information on the project, see Sue McKemmish, Adrian Cunningham and **Dagmar** Parer, 'Metadata Mania: Use of Metadata for Electronic Recordkeeping and Online Resource Discovery' in Place, Interface and Cyberspace: Archives at the Edge, Proceedings of the 1998 Conference of the Australian Society of Archivists, Fremantle 6-8 August 1998. Canberra. Australian Society of Archivists. 1999, **pp 129-144;** and Sue McKemmish and Glenda Acland. 'Accessing Essential Evidence on the Web: Towards an Australian Recordkeeping Metadata Standard.' Paper for AusWeb99 Conference. Available at: **http://ausweb99.scu.edu.au/aw99/papers/mckemmish**. For details of project outcomes, visit the project web site at **http://www.sims.monash.edu.au/rcrg/research/spirt/index.** html

9 Information on the Investing for Growth strategy can be found at: **http://www.dist.gov.au/growth/html/infoage.html**

10 For further information see: **http://law.gov.au/ecommerce/interim3.html**

11 The Project Team, in developing a simple but high level framework model for recordkeeping metadata given as Figure 1, used as an example of effective visual representation the INDECS Community's "Model for Commerce" as derived from David Bearman, Eric Miller, Godfrey Rust, Jennifer Trant and **Stuart** Weibel, 'A Common Model to Support Interoperable Metadata: Progress report on reconciling metadata requirements from the Dublin Core and **INDECS/DOI** Communities.' D-Lib, Vol.5 **No.1,** January 1999. Available at: **http://www.dlib.org/dlib/january99/bearman/01bearman.htm** 1

12 For details of the Australian Government Locator Service see **http://www.naa.gov.au/govserv/agls/)**

13 The Australian Government Locator Service (AGLS) Manual for Users, Version 1.1: 1999-06-09,' Office of Government Online, National Archives of Australia provides details of the application of these types of qualifiers - see **http://www.naa.gov.au/govserv/agls/**

14 Simon Cox has written an excellent discussion paper for the DC community on issues relating to structure, authority and qualification in DC: **http://wwww.agcrc.csiro.au/projects/3018CO/metadata/dc-guide**

15 The Business Acceptable Communications model is described at **http://www.sis.pitt.edu/~nhprc/meta96.html**

16 See **http://www.naa.gov.au/govserv/techpub/rkms/intro.htm**

17 The interactions of the Monash University, Archives and Records Faculty team (Sue McKemmish, Frank Upward, Barbara Reed and Chris Hurley) with David Bearman, particularly at the Managing the Records Continuum intensive workshops in Melbourne and Canberra in June **1996** informed the ideas presented here.

Effective Reuse of Textual Documents Containing Tabular Information

L.E.Hodge, W.A.Gray **and** *N.J. Fiddian*

Department of Computer Science,
Cardiff University,
Cardiff, UK. '

{scmleh|wag|njf}@cs.cf. *ac.uk*

Abstract.

This paper presents an overview of a toolkit that can facilitate efficient reuse of tables appearing within textual documents [1]. In order to effectively reuse information contained in these documents, it is important that we process the accompanying text as well as any tables that appear. From this text, it may be possible to extract metadata such as descriptions of table content and related formulae, mappings and constraints. This metadata can then be exploited to enhance the value of extracted tables during their subsequent reuse. In this paper we present a discussion of the techniques used to process tables and associated text, both of which rely heavily on the use of regular expressions. Our techniques for locating tables utilise similar visual clues used by other table processing techniques discussed in the literature [5,6,7,8], although our approach to exploiting them is quite different. Our tools have been designed to provide a high level of support for the numerous types of table layout encountered in plain text tables, an area that has previously been somewhat overlooked.

1. Introduction.

The growth of the Internet has provided increased access to large bases of textual information that has potential for reuse. Of these documents, some of the most useful (in terms of reuse in information systems) are those that contain information in the form of tables. Tables offer a powerful mechanism for presenting related information in a clear and concise manner. The very nature of tables makes them suitable for reuse since it should be (relatively) straightforward to convert a table from a document into an equivalent table in a spreadsheet or database to facilitate integration with other information systems. Also, the fact that an author has gone to the trouble to present information in a tabular form, would indicate that they felt there was value in presenting the information in this way. This indicates to some extent that the content of the table may have value that can be reused.

In this paper, we focus our discussion on the processing of documents in the form of plain text files (i.e. those that contain only ASCII **text**[1]) as these are the most difficult type of human readable source document to process. Previous work on processing tables in plain text documents has resulted in a number of successful techniques for locating and processing tables. Unfortunately correct understanding of plain text tables has not always been possible using these techniques due to the large variety of layouts that can occur [2]. To overcome these problems, we have developed techniques that utilise the positions of components within the table to determine correct table layout.

Although previously, work has been undertaken to process tables that appear in textual documents, the textual component itself has been ignored. We feel that the text that accompanies tables can be useful in reuse as it often contains information relating to the content of the tables. In [3] we discuss the type of information that may be present in the text along with techniques to faciliate its location and extraction. This information can be employed as a form of metadata to augment any extracted tables and has two basic forms; textual descriptions of table content and technical descriptions i.e. formulae, mappings and constraints.

In this paper, we discuss the techniques we have developed to provide effective processing of both the table and the accompanying text.

[1] In our toolkit, we also support processing of documents that contain embedded mark-up such as HTML and Latex. Processing here is simplified by exploiting embedded tags.

2. Techniques for Processing Tables in Textual Documents.

Most of the techniques for the processing of tables in textual documents that are described in the literature, perform two stages of processing:

i) Locate tables within a document.
ii) Determine the positions of columns and rows within the table and exploit this to determine the layout of the table.

In our toolkit, we add two further processing stages:

iii) Process table content to determine the type of plain text layout occurring - a more powerful technique to determine the layout of the table.
iv) Process accompanying text to locate and extract any available metadata.

In this section, we will examine each of these processing steps in turn.

2.1 Techniques for Locating Tables.

When reading a document, a human locates tables based on their visual form. Computerised processing emulates human behaviour by exploiting the same types of visual clues. The most useful types of visual clue are:

i) Horizontal blocks of white space - commonly used to indicate the boundaries between different components of a document.
ii) Vertical blocks of white space - columns of white space occur between each column of entries in the table and are a good indicator of the presence of a table (see Figure 1).
iii) Horizontal rules - these can also be used to indicate the presence of tables as they are sometimes used to indicate the boundary between the labels and the rest of the table.
iv) Vertical rules - in a fully typeset table, vertical rules may also be used to indicate the edges and column boundaries of a table. Presence of vertical rules is uncommon as they are not essential to understanding and increase the cost of setting a table [2].

```
Name     Age       Sex
--------------------------
Sarah    22        Female
Tim      23        Male
Jo       22        Female
Jon      24        Male
```

Figure 1: A simple plain text table.

Since the inclusion of rules is not mandatory, horizontal and vertical blocks of white space are the only reliable clues in the location of tables. In a number of previous systems [5,6,7,8], a combination of these clues has been used to enable location of tables. Basically, the source document is first broken into blocks by splitting where horizontal blocks of white space appear. Each of these blocks is then processed to check for blocks of vertical white space that could indicate column structure and thus, the presence of a table.

In our system, although we utilise both of these clues, we take a slightly different approach to processing. Initially, we don't split the document and process each block to check if it contains a table. Instead we process the document as a whole and potential tables emerge. We then use the positions of blank lines to check the boundaries of potential tables. Also, we don't perform complex processing on blocks of white space as used in other techniques, but instead employ more simple heuristics based on regular expressions [4] to increase efficiency.

To locate tables, we utilise two different types of heuristic:

i) Double (or more) space count.
ii) Type classification.

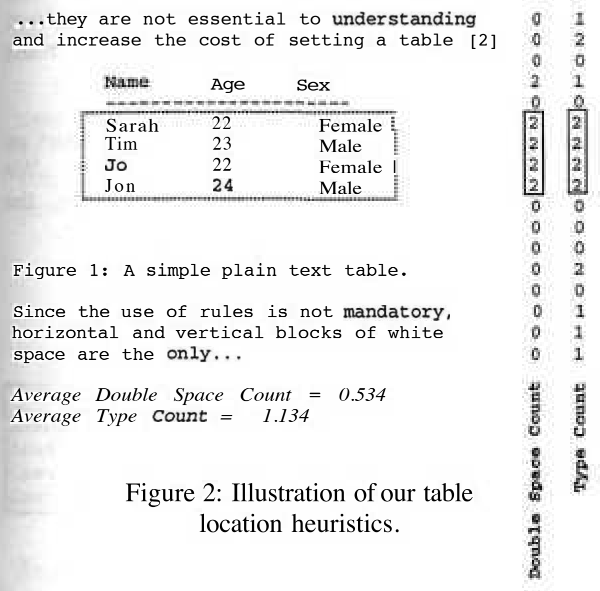
Our system utilises two heuristics to increase recall. Each of the heuristics is best suited to locating a certain class of table types (although both fare reasonably well on their own). Using both heuristics enables our system to locate most, if not all of the tables within a document quickly and efficiently.

Let us now discuss each of these heuristics in turn.

Double (or More) Space Count Heuristic.

This heuristic uses the idea that the columns in most tables are separated by at least two spaces. Thus by searching for such patterns in the text, it should be possible to locate the gaps between columns without extensive processing. We use the regular expression

\b\s\s+\b to match these inter-column gaps[2]. For each line of the source document, we store a count of occurrences of two (or more) spaces. Using these counts, it is possible to locate the positions of potential tables. To do this, a threshold level is set (e.g. the average of the counts) and blocks of (two or more) consecutive lines whose count exceeds this threshold are marked as potential tables. Figure 2 illustrates how tables can be located in this way. The initial block located by this process is illustrated by the dotted rectangle. Notice that if the labels are separated from the body of the table by a horizontal rule (as they are here), the labels will not be included in the block defined by the initial location process. To overcome this problem, if a horizontal rule is located in the line above a potential table block, the top border of the block is moved in order to include the labels.



```
...they are not essential to understanding     0  1
and increase the cost of setting a table [2]   0  2
                                               0  0
      Name        Age       Sex                2  1
      ----------------------------             0  0
      Sarah       22        Female             2  2
      Tim         23        Male               2  2
      Jo          22        Female             2  2
      Jon         24        Male               2  2
                                               0  0
                                               0  0
Figure 1: A simple plain text table.           0  0
                                               0  2
Since the use of rules is not mandatory,       0  0
horizontal and vertical blocks of white        0  1
space are the only...                          0  1
                                               0  1

Average Double Space Count = 0.534
Average Type Count = 1.134
```

Figure 2: Illustration of our table location heuristics.

The Type Classification Heuristic.

This technique exploits the power of regular expressions to act as classifiers for data types. For each line of the source table, we employ a set of regular expressions to divide it into tokens and classify them based on their content (see Figure 3). We support classifiers for string[3], integer, real, float, times and dates of various formats and monies with different currencies.

```
Name  No.  Balance      string
Dave  123  123.45       string,   int,   real
Fred  134  123.56       string,   int,   real
John  145  323.68       string,   int,   real
Carl  166  287.78       string,   int,   real
```

Figure 3: A table with associated type classifications.

As with the double (or more) space heuristic, we store a count of the number of data types that appear in each row. Again, potential tables can be located by locating blocks of lines whose type count exceeds a threshold (as illustrated in Figure 2). Generally, this technique is best at locating entries in the main body of the table. Extra processing using clues such as horizontal rules and blank space is used to locate labels if they are available[4].

2.2 Techniques to Locate Column and Row Positions.

Once a table has been located, it must be processed in order to determine the positions of columns and rows. Clearly, the column structure is indicated by the presence of vertical blocks of white space (streams) between each columns[5]. In order to determine column structure we must exploit these white streams. The most commonly used technique [5,6] is to count the number of white space characters appearing in each column of the text. Peak values will appear where white streams exist and can indicate the positions of column separators. A number of other techniques, mostly based on bounding boxes have also been discussed [7,8].

For our tools, we have developed a technique based on exploiting white streams that is more complex than those commonly used. Instead of simply counting the frequency of space characters in each column of the text, our technique searches for (and stores the size) for the longest continuous stream of white spaces that appear in each column of characters in the block resulting from the table location process. This length count is augmented by a weighting based on the number of non-white space characters that appear in the next column of characters. This indicates the number of entries that will appear in the column generated by this column separator and the next. If relatively few entries will appear in this column, it may not be valid. Thus, it may be incorrect to insert a column separator at this position.

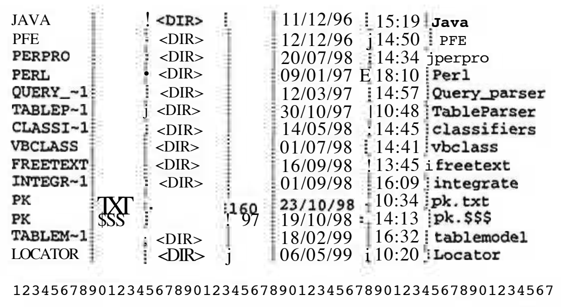
[2] In plain text, a full slop in a passage is usually followed by two spaces. To avoid matching these we use the \b meta-character in the regular expression. Thus reducing the number of matches found by only matching at word boundaries.
[3] To aid processing any stream of letters, spaces and punctuation is classified as a single string.

[4] Again the presence of a horizontal line separates the labels from the body of the table. As before, the top border of the block is moved in order to include the labels.
[5] In some situations it is possible to exploit vertical rules that appear between columns. As these occur infrequently they will not be discussed here.

To locate the positions of column separators, we examine the maximum runs for each column of characters and extract the positions of those that have peak vales[6]. These give an indication of the possible positions of column separators and the weighting acts as an indication of the likelihood that a valid separator has been located.

Consider Figure 4 (an MS-DOS directory listing). If we have a threshold based on the average weighting, we can assign with confidence, four of the six column separators (i.e. those whose weighting exceeds this threshold). Confidence in the remaining separators is low since they don't fit in with the general format of the table. Clearly, to define a column separator at the position of the ninth character in each row would be incorrect as this would split the file names. Even so, we would wish to define a column separator at position 23 in order to define a column for file sizes. To enable this type of processing, two processing modes are required, fully automatic, where all deduced column separators are defined, or semiautomatic where those whose weighting exceed the threshold are automatically set and user interaction is required to decide the validity of any that remain.

To deduce row structure a similar technique is employed to locate horizontal white streams. Because of the complexity of table layout in plain text documents, it is not always possible to determine row structure by exploiting white space alone. To correctly deduce the layout, the positions of the components within the table also need to be considered (as discussed in Section 2.3).



```
JAVA          <DIR>          11/12/96  15:19  Java
PFE           <DIR>          12/12/96  14:50  PFE
PERPRO        <DIR>          20/07/98  14:34  jperpro
PERL          <DIR>          09/01/97  18:10  Perl
QUERY_~1      <DIR>          12/03/97  14:57  Query_parser
TABLEP~1      <DIR>          30/10/97  10:48  TableParser
CLASSI~1      <DIR>          14/05/98  14:45  classifiers
VBCLASS       <DIR>          01/07/98  14:41  vbclass
FREETEXT      <DIR>          16/09/98  13:45  freetext
INTEGR~1      <DIR>          01/09/98  16:09  integrate
PK       TXT            160  23/10/98  10:34  pk.txt
PK       $$$             97  19/10/98  14:13  pk.$$$
TABLEM~1      <DIR>          18/02/99  16:32  tablemodel
LOCATOR       <DIR>          06/05/99  10:20  Locator

1234567890123456789012345678901234567890123456789012345678901234567
```

```
Max runs  found at:
Position 9 with weighting 2
 "    "   15 with weighting 12
 "    "   23 with weighting 2
 *    "   28 with weighting 14
 "     " 3 8 with weighting 14
 *    "   44 with weighting 14

Average weighting is:  9.67
```

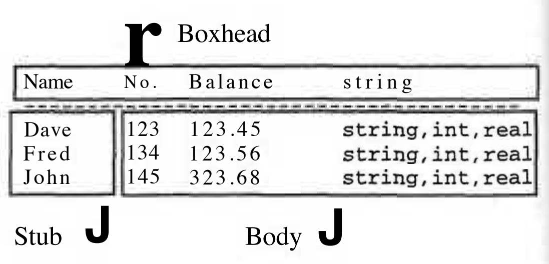
Figure 4. Illustration of technique for locating column separators.

[6] Where a number of neighbouring columns have the same peak value, the last one in the group is chosen.

2.3 Techniques for Processing Table Content.

In most of the existing table processing systems, the techniques used to determine the layout of data in the table are fairly simple. In most cases, only limited processing is undertaken to determine the formatting of rows (based again on white space). Determining column structure has been more extensively investigated and multiple layers of labelling are often supported. Unfortunately, in most cases, the layout of tables is assumed to be fairly simple. Alas, when working with plain text tables, this is not always the case. The flexibility of layout offered by plain text results in a large variety of different types of table format. Many of which are not suitable for direct reuse. Previously little emphasis has been placed on providing support for the different types of layout that occur. In our investigation we have focused on the processing that would be required to transform source tables into a relational form, as this is the most suitable form for reuse[7].

In our toolkit we have attempted to support as many different types of layout as possible. We classify these types of layout based on the components of the table as illustrated in Figure 5.



Name	No.	Balance	string
Dave	123	123.45	string,int,real
Fred	134	123.56	string,int,real
John	145	323.68	string,int,real

Figure 5: Components of a table.

Within each of these components we see different types of layout. The boxhead and the stub sections offer the author most control over the way that data is presented in the table and correct processing of these components is vital to correct reuse of table contents[8]. For each component, let us now briefly describe the types of layout that occur.

Boxhead Layouts.

The simplest layout in the boxhead consists of a single layer of labels (see Figure 1). This type of boxhead is easy to handle, as no extra processing is required to reuse its content. The appearnce of multiple levels of labelling is also common in the boxhead (Figure 6). Here, grouping of related columns is possible, making the table content more self explanatory. Where multiple levels of labelling occur, the boxhead will need to be processed to deduce a single layer of labels. This is achieved by creating a simple compound label based on a downward traversal through the appropriate labels (i.e. Examinations.Mid-term and Examinations.Finals).

[7] Relational tables can be easily imported into most databases and spreadsheets. Techniques to support integration with other types of DBMS are well understood.

[8] The layout of the table body also affects the processing, although not to the same degree as the boxhead and stub.

	Examinations			
Name	Mid-term	Finals	Total_Mark	Grade
T.Thome	70	71	70.5	A
S.Warrilow	65	70	67.5	B
J.Bloggs	64	78	71.0	A

Figure 6: A table with multiple layers of labelling in the boxhead.

In some cases (usually due to space constraints or appearance), labels may span multiple lines (Figure 7). Where this occurs it is important that our tools are able to determine the type of processing required and generate appropriate labels[9].

	1900	
	% of	Seats
Party	vote	won
xxxxxxx	xx	xxxxx

Figure 7: An example of a boxhead containing multi-level labelling.

Stub Layouts.

The simplest type of stub (as in Figure 6) requires no processing as it consists of a single column of entries. Stubs in plain text tables also support a type of classification hierarchy through the use of nested entries (Figure 8). This allows elimination of repeated labelling resulting in a more compact table. Where stubs of this kind are encountered, they must be expanded into multiple columns (one for each level of indent) and repeated entries reintroduced (Figure 9). Notice that stubs with nested entries rarely have a label in the boxhead. Thus, when a stub of this kind is expanded, appropriate labels will need to be inserted by the user.

[9] Due to space constraints we are unable to discuss processing of multi-line entries in this paper.

	Price
Ford	
Fiesta	
1.1	8000
1.3	8650
Vauxhall	
Astra	
1.3	8500
1.6	9250
2.0	12000

Figure 8: A table with a stub containing nested entries.

Maker	Model	E_Size	Price
Ford	Fiesta	1.1	8000
Ford	Fiesta	1.3	8650
Vauxhall	Astra	1.3	8500
Vauxhall	Astra	1.6	9250
Vauxhall	Astra	2.0	12000

Figure 9: The result of processing the table in Figure 8.

Again, stub entries may span over multiple lines. Where this occurs, it is important that the entry is handled as a single entity. Where multi-line stub entries occur, it is essential that some visual clues can be located to allow correct processing (e.g. blank lines or rules between each row).

Body Layouts.

The simplest body layout is identical to a relational table and requires no special processing (as in Figure 4). Problems can sometimes occur where entries in the body span multiple lines (Figure 10). Where this occurs, correct processing relies on visual clues as it does for multi-line stub entries.

Model	Description
Pro-Yo II	Wooden axle, beginner or intermediate yoyo
RB II	Roller bearing transaxle yoyo. Good looper. intermediate / Expert use.
Viper	Roller bearing transaxle yoyo. Butterfly shape good for string tricks. Rubber sides !

Figure 10: A table with multi-line entries in the body.

2.4 Techniques for Extracting Metadata from Accompanying Text.

From the text that accompanies tables, we support the extraction of four types of metadata:

i) Semantic descriptions - textual descriptions related to the content of columns in the table.
ii) Formulae - that define how a column may be materialised from others e.g. Total_Mark = (Mid-term + Finals)/2.
iii) Mappings - that define how a column may be materialised from others e.g. if Total_Mark >= 70 then Grade = 'A'.
iv) Constraints - that define the range of valid entries for a column e.g. Total_Mark >= 0 and <=100.

For all types of metadata our extraction techniques utilise regular expressions. In simple terms, processing involves dividing the text into sentences, then checking to see if any metadata appears in these sentences by matching using a regular expression.

Regular expression to match a formula.

```
(\w*\s*=\s*((\(*|\)*)\s*)?(-?([0-9]+(\.
[0-9]*)?|\.[09]+)|\w*)\s*((\(*|\)*)\s*)?
(\+|-|\/|\*|\^)\s*((\(|\))\s*)?(-?([0-
9]+(\.[0-9]*)?|\.[0-9]+)|\w*)((\s*((\
(|\))\s*)?(?([0-9]+(\.[0-9]*)?|\.[0-9]+)
|\w*)(\s*((\(|\))\s*)?(\+||\/|\*|\^)\s*
((\(|\) )\s*)?\w*(\s*(\(|\) ) \s*)?)*)?) (\s
*(\(|\))\s*)?)
```

Regular expression to match a mapping .

```
(if(\s*)\w*(\s*)?(\<|\>|=|\<\>|\<=|\>=)(
\s*)\d*(\s*)?(and(\s*)?(\<|\>|=|\<\>|\<=
\>=)(\s*)?\d*(\s*)?)?then(\s*)?\w*(\s*)?
=\s*((\(*|\)*)\s*)?(-?([0-9]+(\.[0-]*)?
|\.[0-9]+)|\w*)\s*((\(*|\)*)\s*)?(\+|-
|\/|\*)\s*((\(|\))\s*)?(-?([0-9]+(\.[0-
9]*)?|\.[0-9]+)|\w*)((\s*((\(|\))\s*)?(-
?([0-9]+(\.[0-9]*)?|\.[0-9]+)|\w*)(\s*
((\(|\))\s*)?(\+||\/|\*)\s*((\(|\))\s*)?
\w*(\s*(\(|\))\s*)?)*)?)(\s*(\(|\))\s*)?
)
```

Regular expression to match a constraint .

```
(\w*(\s*)?(\<|\>|=|\<\>|\<=|\>=)(\s*)\d*
(\s*)?and(\s*)?(\<|\>|=|\<\>|\<=|\>=)(\s
*)?\d*)
```

Figure 11: The regular expressions used to locate formulae, mappings and constraints.

For semantic descriptions this simply involves searching for column labels in the text. If they appear, the appropriate sentences are extracted and are used to build a type of key or dictionary. This can then be used by the end user to get a quick overview of the content and meaning of the table without reading all of the accompanying text. For the more technical metadata, we utilise the regular expressions shown in Figure 11. Where successful matches are made, this metadata is stored for reuse in the destination application.

3. Reuse of Table Content and Metadata.

In order to demonstrate how useful content extracted from documents can be reused, our toolkit provides facilities to export tables and associated metadata into two types of application - an Excel spreadsheet and an INGRES database.

We support reuse in Excel via the Visual Basic for Applications API. This allows our tools to exploit the functionality of Excel to generate appropriate spreadsheets with appropriate formulae, mappings and constraints. Semantic descriptions are used to generate pop up descriptions attached to the label for each column, allowing the user a brief description of the content of each column.

To support generation of an INGRES database, our tools generate appropriate SQL to set up and populate a table[10] along with appropriate rules and procedures to support formulae, mappings and constraints. Obviously we can not utilise semantic descriptions directly in INGRES so these are made available to the user in a text file.

4. Conclusion.

In this paper we have presented an overview of a toolkit to facilitate the effective reuse of textual documents that contain data in a tabular form. Whereas previous table processing systems have concentrated solely on the location and processing of tables, our tools exploit the accompanying text to extract any metadata that may be available. This metadata is then used to add value to the extracted tables.

We have specifically designed tools to support the numerous different table layouts that are available in plain text documents. By developing techniques for the different types of layout that occur in different table components we are able to achieve a high level of precision and recall.

Although in this paper, we have concentrated our discussion on plain text source documents, our toolkit also supports textual documents that contain embedded mark-up, such as HTML and Latex. In providing support for documents of this type (i.e. document definition languages DDL's) we have utilised a meta-language approach [9]. Using this approach allows simple extension of our processing to support any additional DDL's should the need arise.

[10] We use a system or regular expressions to determine the data type for each column in order to define the table schema.

References.

[1] L. E. Hodge, W. A. Gray and N. J.Fiddian. A Toolkit to Facilitate the Querying and Integration of Tabular data from Semi-structured Documents. In *Proceedings of the 16th British National Conference on Databases, (BNCOD 16),* July 1998.

[2] *The Chicago Manual of Style,* Thirteenth Edition, The University of Chicago Press, 1982.

[3]L. E. Hodge, W. A. Gray and N. J. Fiddian. Deduction of Metadata for Tabular Structures in Plain Text. In *Proceedings of IEE Conference on Multimedia Databases and MPEG-7,* 1999.

[4] J. Friedl. *Mastering Regular Expressions,* O'Reilly and Associates, 1997.

[5]S. Chandran and R. Kasturi. Structural Recognition of Tabulated Data. In *Proceedings of the International Conference on Document Processing (ICDAR 93),* 1993.

[6] K. Itonori. Table Structure Recognition Based on Textblock Arrangement and Ruled Line Position. In *Proceedings of the International Conference on Document Processing (ICDAR 93),* 1993.

[7]M. A. Rahgozar, Z. Fan and E. V. Rainero. Tabular Document Recognition. In *Proceedings of the SPIE Conference on Document Recognition,* 1994.

[8]T. G. Kieninger. Table Structure Recognition Based on Robust Block Segmentation. In *Proceedings of Electronic Imaging 98 (SPIE), Document Recognition,* 1998.

[9] D. I. Howells. *A Source-to-Source Meta-translation System for Database Query Languages.* Ph.D. Thesis, Cardiff University, 1988.