Representing and Rendering Linguistic Paradigms

David Penton and Steven Bird
Department of Computer Science and Software Engineering
The University of Melbourne, Victoria 3010, Australia
{djpenton,sb}@csse.unimelb.edu.au

Abstract

Linguistic forms are inherently multi-dimensional. They exhibit a variety of phonological, orthographic, morphosyntactic, semantic and pragmatic properties. Accordingly, linguistic analysis involves multi-dimensional exploration, a process in which the same collection of forms is laid out in many ways until clear patterns emerge. Equally, language documentation usually contains tabulations of linguistic forms to illustrate systematic patterns and variations. In all such cases, multi-dimensional data is projected onto a two-dimensional table known as a linguistic paradigm, the most widespread format for linguistic data presentation. In this paper we develop an XML data model for linguistic paradigms, and show how XSL transforms can render them. We describe a high-level interface which gives linguists flexible, high-level control of paradigm layout. The work provides a simple, general, and extensible model for the preservation and access of linguistic data.

1 Introduction

A linguistic paradigm is any kind of rational tabulation of linguistic forms, such as phrases, words, or phonemes, intended to illustrate contrasts and systematic variation (Bird, 1999). A characteristic property of paradigms is that interchanging entire rows or columns does not change the interpretation of the information. We view paradigms as a two-dimensional arrangement of elements and attributes, with optional row and column labels. An example of a paradigm for the German definite article is shown in Figure 1, with the labelling of number and gender at the top, and case on the left.

This paper describes a relational data model for linguistic paradigms, together with an XML based approach for representing and rendering them. An XSLT implementation provides proof of concept.[1] This work presents a simple and general model for the preservation and access of linguistic paradigms, and can generate an extensive range of useful visualisations.

PARADIGM FOR GERMAN DEFINITE ARTICLE

	SINGULAR			PLURAL
	MASCULINE	FEMININE	NEUTER	ALL GENDERS
NOMINATIVE	der	die	das	die
ACCUSATIVE	den	die	das	die
GENITIVE	des	der	des	der
DATIVE	dem	der	dem	den

Figure 1: Paradigm for German definite article (Finegan, 1999, 60)

This paper is organised as follows. In §2 we discuss the existing computational models in linguistic paradigms and the lack of an existing formalism. In §3 we discuss the data model, while §4 and §5 we provide an example of how to generate and visualise a linguistic paradigm. In §6 we discuss the query engine implementation and motivate each operation. In §7 we describe the transformation of the paradigm data into an XHTML presentation form. Finally, §8 discusses the significance of the work and outlines several areas for further investigation.

2 Background

Traditionally, there have been two sources for computational representations for linguistic paradigms, descriptive linguistic tools and technologies for languages having complex morphology. There has been little formal work in either area concerning the best form for this linguistic data type. Here we examine some of the more widely-used models and their drawbacks.

Of the descriptive linguistic tools, perhaps the foremost are Shoebox[2] and CHILDES.[3] Shoebox is an interlinear text editor popular among field linguists for analysing linguistic transcripts. The underlying model is an attribute value set for each

[1] The implementation is available from http://www.csse.unimelb.edu.au/research/lt/projects/paradigms/

[2] http://www.sil.org/computing/shoebox/

[3] http://childes.psy.cmu.edu/

Proceedings of the Australasian Language Technology Workshop 2004
Macquarie University, Sydney, December 8th, 2004.

Accepted after full paper review

Tag	Value	Description
\id	1612	identifier (used for hyperlinks)
\w	mbhū	orthographic form
\t	LDH	tone transcription
\p	n	part of speech
\pl	me-	plural prefix
\cl	9/6	noun class (singular/plural)
\en	dog	English gloss
\fr	chien	French gloss

Figure 2: Shoebox File Format, Adapted for Linguistic Paradigms (Bird, 1997)

entry, as shown in Figure 2. In the context of paradigms, each element corresponds to a cell in a table, and Shoebox can generate simple tabulated listings of forms which constitute a one-dimensional paradigm. The CHILDES CLAN tool supports transcription and analysis of conversation, and is widely used by psycholinguists in their study of child language acquisition. It has good search functionality that permits the generation of tabular reports. Despite their ability to generate simple paradigm-like reports, these systems do not provide an interface for generating arbitrary paradigms, nor do they permit paradigms to be saved in a format which permits reuse.

Outside purely linguistic description, work on computational morphology usually requires paradigms to be set up. For instance, Finnish and Romanian have such a large number of productive morphological processes it is impractical to list every form in the lexicon. Instead, regular derivational and inflectional processes are described using a formal system (such as a finite-state transducer). Groups of processes which apply to the same class of lexical items are sometimes referred to as a paradigm (e.g. (Tufis, 1989; Oflazer et al., 2001)). Unlike the descriptive viewpoint, in which a paradigm is a tabulation, here a paradigm is effectively treated as executable code which might be used to generate such tabulations. However, we are neutral on this issue since both viewpoints can be reconciled by treating a paradigm as a relation, as we do in §3.

3 Data Model

Linguistic paradigms associate linguistic forms with linguistic categories. For instance, the German definite article paradigm in Figure 1 categorises the form *den* as either masculine singular accusative or as dative plural. Systematic changes in layout, such as interchanging rows and columns, or flipping axes, have no affect on the associations between

D_1 gender	D_2 number	D_3 case	D_0 content
masc	sg	nom	der
masc	sg	acc	den
masc	sg	gen	des
masc	sg	dat	dem
masc	pl	nom	die
masc	pl	acc	die
masc	pl	gen	der
masc	pl	dat	den
fem	sg	nom	die
fem	sg	acc	die
fem	sg	gen	der
fem	sg	dat	der
fem	pl	nom	die
fem	pl	acc	die
fem	pl	gen	der
fem	pl	dat	den
neut	sg	nom	das
neut	sg	acc	das
neut	sg	gen	des
neut	sg	dat	dem
neut	pl	nom	die
neut	pl	acc	die
neut	pl	gen	der
neut	pl	dat	den

Figure 3: Function for the German Paradigm

forms and categories. Accordingly, a paradigm is a function that maps a vector of properties to content, as follows:

$$f : \langle \text{masc}, \text{sg}, \text{acc} \rangle \mapsto \text{den}$$

Generalising, let $D_0 \ldots D_n$ be a set of linguistic properties (or domains). Then a paradigm is a function:

$$f : D_1 \times \cdots \times D_n \rightarrow D_0$$

Let $D_1 = \{\text{masc}, \text{fem}, \text{neut}\}$, $D_2 = \{\text{sg}, \text{pl}\}$, and $D_3 = \{\text{nom}, \text{acc}, \text{gen}, \text{dat}\}$. Also, let $D_0 = \{\text{der}, \text{die}, \text{das}, \ldots\}$. The functional representation of the German paradigm is shown in Figure 3.

Observe that the original paradigm display in Figure 1 is a compact view of this table. It shows the domain values just once, and dispenses with the gender property for the plural forms.

Now, the above functional representation in Figure 1 is just a relational table with schema GermanParadigm (gender, number, case, content). We can use domain relational calculus to extract the columns of the original paradigm for display, e.g.:

$$\{s \mid t \in \text{GermanParadigm} \wedge t[\text{number}] = \text{`sg'}$$

Accepted after full paper review

$$\wedge\, t[\text{gender}] = \text{`masc'} \wedge t[\text{case}] = s[\text{case}]$$
$$\wedge\, t[\text{content}] = s[\text{content}]\}$$
$$= \{\langle \text{nom}, \text{der}\rangle, \langle \text{acc}, \text{den}\rangle, \langle \text{gen}, \text{des}\rangle, \langle \text{dat}, \text{dem}\rangle\}$$

The same query is expressed in SQL as follows:

```
SELECT case, content
FROM GermanParadigm
WHERE number = "sg"
AND gender = "masc".

nom, der
acc, den
gen, des
dat, dem
```

Standard XML technologies provide a more convenient way to map from this abstract representation to a range of visualisations. The relational table representation of a paradigm in XML is as follows:

```
<paradigm>
  <form>
    <attribute name="gender" value="masc"/>
    <attribute name="number" value="sg"/>
    <attribute name="case" value="nom"/>
    <attribute name="content" value="der"/>
  </form>
  ...
</paradigm>
```

XSL transforms provide a method to render this material into XHTML or into some other presentational markup language for delivery to users. Using this approach we will accomplish a round-trip, from existing visualisations to the abstract model discussed here, then back to visualisations. The next two sections describe this process in more detail.

4 Generating Paradigms

This section provides an overview of the steps required to produce a paradigm of possessive pronouns in Anejom̃, an Austronesian language of Vanuatu. The source data is from (Lynch, 1998). The process of generating paradigms is the same for paradigms that are more complicated, so the simplified Anejom̃ paradigm provides a helpful introduction. First there is a simple examination of the paradigm structure which motivates the choice of model for the paradigm. Then, by investigating an example query and looking at how it is processed and presented, the intricacies and obstacles to an effective model are evident. This leads onto the discussion in §6 and §7 which provides the finer details of the implementation and presents more elaborate examples in order to reveal the strengths and weaknesses of the model.

Figure 4 shows the architecture of the system. The processing pipeline has three components: an XML model, a query engine, and a presentation engine.

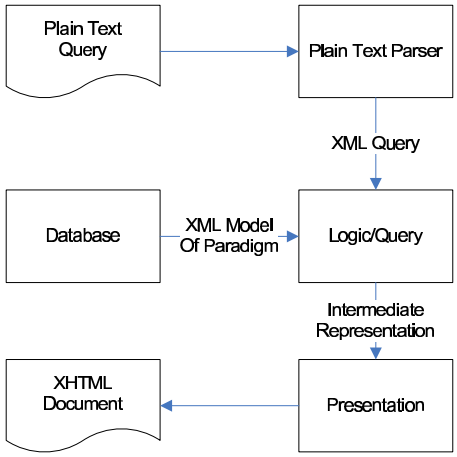

Figure 4: Architecture of System

Table 5. Anejom̃ Pronouns

	1. INC	1. EXC	2.	3.
Possessive				
Singular	—	*-k*	*-m̃*	*-n*
Dual	*-jau*	*-mrau*	*-mirau*	*-rau*
Trial	*-taj*	*-mtaj*	*-mitaj*	*-ttaj*
Plural	*-ja*	*-ma*	*-mia*	*-ra*

Figure 5: Anejom̃ Possessive Pronouns (Lynch 1998:106) - Scanned Version

Figure 5 shows a visualisation of a paradigm for Anejom̃. It displays suffix morphemes for possessive pronouns for different combinations of number and person. Each cell is characterised by its content and its attributes. For instance, the top right cell has content *-n*, a 'number' attribute whose value is 'singular', and a 'person' attribute whose value is '3'. Each attribute has a domain of possible values. For example, the domain of 'number' is 'singular', 'dual', 'trial' and 'plural'. The content is likewise a domain, having values such as '-jau' and '-mrau'. Figure 6 shows the XML model for Anejom̃. The attributes and their domains make up the first section, while the cells – the correspondences between content and attribute values, make up the second section.

The XML model provides a representation for the paradigm; the remainder of this section describes a plain text query language for generating different presentations from that model. The plain text query maps to an XML based representation. Then a XSL transform performs the underlying operations on the

Accepted after full paper review

```
<?xml version="1.0" encoding="UTF-8"?>
<document>
  <attributes>
    <name name="person">
      <value value="1.INC"/>
      <value value="1.EXC"/>
      <value value="2"/>
      <value value="3"/>
    </name>
    <name name="number">
      <value value="singular"/>
..
      <value value="plural"/>
    </name>
    <name name="content">
      <value value="-"/>
      <value value="-jau"/>
      ..
      <value value="-ra"/>
    </name>
  </attributes>

  <paradigm>
    <form>
      <attribute name="person" value="1.INC"/>
      <attribute name="number" value="singular"/>
      <attribute name="content" value="-"/>
    </form>
    <form>
      <attribute name="person" value="1.INC"/>
      <attribute name="number" value="dual"/>
      <attribute name="content" value="-jau"/>
    </form>
    ..
    <form>
      <attribute name="person" value="3"/>
      <attribute name="number" value="plural"/>
      <attribute name="content" value="-ra"/>
    </form>
  </paradigm>
</document>
```

Figure 6: XML Model of Anejom̃ Possessive Pronouns

XML model of the paradigm. Here is an example. The Query 1 produces the visualisation of Figure 7 from the XML model of Figure 6. Translation 1 shows the full query. The table operator takes three arguments, the constraint applied to the vertical axis, the constraint applied to the horizontal axis, and the operation applied in each cell. The domain operation presents a list of all the values in a given domain. Note that the domain for the third argument of a table operation is different for each cell and determined by the values on the vertical and horizontal axes. Therefore Figure 7 shows a table with person and number as axes and content in the cells.

Query 1: table(person, number, content)

Translation 1: table(domain(person), domain(number), domain(content))

	1.INC	1.EXC	2	3
singular	-	-k	-m	-n
dual	-jau	-mrau	-mirau	-rau
trial	-taj	-mtaj	-mitaj	-ttaj
plural	-ja	-ma	-mia	-ra

Figure 7: Anejom̃ Possessive Pronouns: Table - Reproduced Visualisation

1.INC	singular	-
	dual	-jau
	trial	-taj
	plural	-ja
1.EXC	singular	-k
	dual	-mrau
	trial	-mtaj
	plural	-ma
2	singular	-m
	dual	-mirau
	trial	-mitaj
	plural	-mia
3	singular	-n
	dual	-rau
	trial	-ttaj
	plural	-ra

Figure 8: Anejom̃ Possessive Pronouns - Hierarchy induced by query

The model supports multiple visualisations of the data. For example, Query 2 produces a presentation of the XML model in a tree-like structure. In Query 2 the shorthand '/' symbol represents the hierarchy operation shown in Translation 2. The hierarchy operator takes two arguments, the constraint that forms a list and the operation applied to each element of the list. This produces the visualisation of Figure 8. Nesting table and hierarchy operations allows presentation of paradigms that are more complicated and n-dimensional.

Query 2: person/number/content

Translation 2: hierarchy(domain(person), hierarchy(domain(number), domain(content)))

5 Realising Paradigms

This section details the implementation responsible for presenting queries, using our running example. First, a PHP script maps the textual query representation to the equivalent XML representation shown in Figure 9. Then the 'logical' transform runs the query on the underlying XML model of the Anejom̃ paradigm (See Figure 6). Finally, the 'presentational' transform generates an XHTML presentation of that paradigm. Both transforms are written using XSLT.

Accepted after full paper review

```
<?xml version="1.0"?>
<document>
  <parse-tree>
    <operator opcode="table" instruction="1">
      <operand type="domain"
        arg="horizontal">person</operand>
      <operand type="domain"
        arg="vertical">number</operand>
      <operand type="domain"
        arg="cell">content</operand>
    </operator>
  </parse-tree>
</document>
```

Figure 9: XML Version of Query 1

The logical transform generates an intermediate representation from the XML query and the XML source model. The XSLT processor performs a depth first traversal of the query expression. For example, in Query 1 control starts at the table operation. The table operation requires calculating the domain of person and number before it can generate the cells. The domain operation generates the output tree of Figure 10 with a node for each value in its domain. The processor generates nodes 1INC, 1EXC, 2 and 3 for person. It then places the forms from the source tree that match the domain value under the corresponding output node.

When processing the table operation the XSLT processor searches the output trees of the vertical and horizontal branches for child nodes. The XSLT processor generates a cell for each combination of vertical horizontal child pairs. The combined set of nodes for each cell form the domain of the third argument. In the example, the first cell of the paradigm has the following mapping:

vertical: singular {-, -k, -m. -n}
horizontal: 1.INC{-, -jau, -taj, -ja}
cell(1,1): {-}

```
<form>
  <attribute name="person" value="1.INC"/>
  <attribute name="number" value="singular"/>
  <attribute name="content" value="-"/>
</form>
```

The query domain(content) in cell (1,1) produces a single node with a value '-'. The extended XML output of the sheet is shown in Figure 10. At each node the XSLT processor tags the number of leaves and maximum depth of the tree which simplifies the presentation logic.

The presentational transform renders the intermediate representation into XHTML for display on web browsers. It traverses the

```
<?xml version="1.0"?>
<document><operator optype="table">
  <vertical leaf-depth="1" leaves="4">
    <operator optype="domain" root-depth="1"
leaf-depth="1" leaves="4" direction="top-to-b.">
      <node ..>
        <att><html-att. element-name="th"/></att>
        <forms>
          <form>
            <att. name="person" value="1.INC"/>
            <att. name="number" value="singular"/>
            <att. name="content" value="-"/>
          </form>
          <form>
            <att. name="person" value="1.EXC"/>
            <att. name="number" value="singular"/>
            <att. name="content" value="-k"/>
          </form>
          <form>
            <att. name="person" value="2"/>
            <att. name="number" value="singular"/>
            <att. name="content" value="-m"/>
          </form>
          <form>
            <att. name="person" value="3"/>
            <att. name="number" value="singular"/>
            <att. name="content" value="-n"/>
          </form>
        </forms>
      </node> . <node heading="plural" ../>
    </operator>
  </vertical>
  <horizontal leaf-depth="1" leaves="4">
    <operator optype="domain" root-depth="1"
leaf-depth="1" leaves="4" direction="left-to-r.">
      <node heading="1.INC" ..>
        <forms>
          <form>
            <att. name="person" value="1.INC"/>
            <att. name="number" value="singular"/>
            <att. name="content" value="-"/>
          </form>
          <form>
            <att. name="person" value="1.INC"/>
            <att. name="number" value="dual"/>
            <att. name="content" value="-jau"/>
          </form>
          <form>
            <att. name="person" value="1.INC"/>
            <att. name="number" value="trial"/>
            <att. name="content" value="-taj"/>
          </form>
          <form>
            <att. name="person" value="1.INC"/>
            <att. name="number" value="plural"/>
            <att. name="content" value="-ja"/>
          </form>
        </forms>
      </node> . <node heading="3" ../>
    </operator>
  </horizontal>
  <cells>
    <row>
      <column>
        <operator optype="domain" root-depth="1"
leaf-depth="1" leaves="1" direction="left-to-r.">
          <node heading="-" ..>
            <forms>
              <form>
              <att. name="person" value="1.INC"/>
              <att. name="number" value="singular"/>
              <att. name="content" value="-"/>
            </form>
            </forms>
          </node>
        </operator>
      </column><column>..</column>
    </row><row>..</row>
  </cells>
</operator></document>
```

Figure 10: XML output from Query 1. The forms elements are only relevant in the horizontal and vertical nodes during processing. The common form element becomes the form element for the cell.

Accepted after full paper review

intermediate representation depth-first from the root. The XSLT document handles three classes of node; text leaves (node), text nodes with children (domains) and arrays of text nodes with children (tables). It must handle the following cases for each node; where the node is the contents of another node; and where the orientation of the node is vertical or horizontal. The next section provides further detail of how the query engine presents paradigms.

6 Operations

This section elaborates the domain, hierarchy and table operations, showing how they describe the presentation of a wide variety of linguistic paradigms. We also provide correspondences with relational queries:

Query 3: person

Translation 3: domain(person)

The SQL equivalent of Translation 3 is as follows:

```
SELECT person FROM paradigm
```

person
1.INC
1.EXC
2
3

Query 4: person/number

Translation 4: hierarchy (domain(person), domain (number))

```
SELECT * FROM
  (
    SELECT person FROM paradigm
      OUTER JOIN
    SELECT number FROM paradigm
  );
```

person	number
1.inc	singular
	dual
	trial
	plural
1.exc	singular
	dual
.	.
.	.

Query 5: table(person, number, content)

Translation 5: table(domain (person), domain (number), domain (content))

```
Qv = SELECT person FROM paradigm;
Qh = SELECT number FROM paradigm;
Qd = Qv OUTER JOIN Qh OUTER JOIN paradigm;
Qc = SELECT content FROM Qd;
```

The domain and hierarchy queries have straightforward mappings to SQL as shown for Translation 3 and Translation 4. The interactions of the table operation are complex, especially when looking at the query that produces the cells. To build the table the parser generates the axes using the query for the vertical and horizontal axes with a direct mapping of the queries from the first two arguments as shown in Translation 5. In this case, it is the queries for domain(person) and domain(number) that produce the desired SQL. A projection of the vertical and horizontal values (`Qv` and `Qc`) form the domain (`Qd`) for each of the cells (`Qc`). Any query on the cells applies only to this new domain (`Qd`). The result of the query `Qc` is a list of values which fill the table from the top left corner.

	Masculine Nouns	*Feminine Nouns*
Singular	*le bon livre* 'the good book'	*la bonne maison* 'the good house'
	ce livre vert 'this green book'	*cette maison verte* 'this green house'
	mon grand frère 'my big brother'	*ma grande soeur* 'my big sister'
Plural	*les bons livres* 'the good books'	*les bonnes maisons* 'the good houses'
	ces livres verts 'these green books'	*ces maisons vertes* 'these green houses'
	mes grands frères 'my big brothers'	*mes grandes soeurs* 'my big sisters'

Figure 11: French concord (Crowley, 1992, 322)

This covers the simple case where the paradigm has just three-dimensions. There are however, paradigms that have many more dimensions. The French concord of Figure 11 has five dimensions and provides a good source for more complicated queries. Consider the following two queries:

Query 6: table (gender, number, language/phrase)

Translation 6: table(domain(gender), domain(number), domain(language/phrase))

```
Qv = SELECT gender FROM paradigm
Qh = SELECT number FROM paradigm
Qd = Qv OUTER JOIN Qh OUTER JOIN paradigm
Qc =
  SELECT * FROM (
    SELECT language FROM Qd
    OUTER JOIN
    SELECT phrase FROM Qd
);
```


Accepted after full paper review

Query 7: table(gender, number, language)

Translation 7: table(domain(gender), domain(number), domain(language))

```
Qd = paradigm
Qc =
SELECT * FROM (
  SELECT language FROM Qd
  OUTER JOIN
  SELECT phrase FROM Qd
);
```

The difference in SQL for the cells between these two queries is the domain `Qd`. This represents the context for the query, its treatment is systematic throughout the XSLT logic allowing nesting of queries. Query 8 and Query 9 show queries that produce two different structures for the French language data. Using combinations of domain, hierarchy and table operation it is possible to generate almost all presentation layouts.

Query 8: table(gender/number, case/language, phrase)

Translation 8: table(hierarchy(domain(gender), domain(number)), hierarchy(domain(case), domain(language)), domain(phrase))

Query 9: table(gender, case, table (number, language, phrase))

Translation 9: table(domain(gender), domain(case), table(domain(number), domain(language), domain(phrase))

7 Presentation

This section describes the implementation of the presentation engine. This is the most complex component in the system; it produces XHTML from the underlying XML representation. The XSL transform does the processing with each operation acting as event. The domain operation is the simplest operation. It handles three distinct cases; one case for producing a list of values; one for producing a horizontal table; and one for producing a vertical table. The processing produces the following code for each:

```
Row:
<tr><td>Item 1</td><td>Item 2</td></tr>

Column:
<tr><td>Item 1</td></tr>
<tr><td>Item 2</td></tr>

Space separated list:
Item 1 Item 2
```

XSLT recursion solves the more difficult problem of constructing a hierarchy. The example of Table 1

Item 11		Item 12	
Item 21	Item 22	Item 23	Item 24

Table 1: Horizontal hierarchy of items.

Item 11	Item 21
	Item 22
Item 21	Item 23
	Item 24

Table 2: Vertical hierarchy of items.

is straightforward; each node has width equal to the number of its children (set with the colspan property). When the hierarchy is root, each level is a row with control grounded at the root. Control must be grounded at the root to avoid parts of the tree ending up in different rows. In Table 1 this equates to 'Item 11' and 'Item 12' forming one row and 'Item 21', 'Item 22', 'Item 23' and 'Item 24' forming the second row.

The same is true for the vertical realisation: the root node controls the generation of each row. However, in this case, there is a need for a policy for when to generate XHTML nodes. The problem is the XHTML language has one nesting of row and column yet two directions of spanning cells. Thus, in Table 2, 'Item 11' produces a node in row one but not row two and 'Item 21' produces a node in row three but not row four. This causes serious difficulties for any program written in a functional language. The solution is an intricate variable passing procedure where the generation of each label depends on whether it is the first node in the row. When it is first the label forms a cell with the rowspan property equal to the number of children in the hierarchy.

The generation of tables comes in three parts; the generation of the vertical axis; the horizontal axis and the cells. The XSL transform leaves the top-left square of the paradigm blank to avoid connection ambiguity problems. The generation of the horizontal axis is the same as when the table did not exist, albeit appropriately shifted by the depth of the vertical axis. The table operator iterates over each row generating first the vertical heading for that row then the cells for that row. This maps to the XHTML design of the table where the declaration of the rows comes before the columns.

The XSL transform treats operators as either controllers or fillers. As controller, the operator has responsibility for generating XHTML table and row tags. As filler, the operator just has responsibility for generating content. The nature of different orientations require different code for vertical and

horizontal orientations. When supported this allows integration and presentation of arbitrary commands. In fact this XSLT framework can display any query that used operations from §6.

8 Conclusion

This paper describes an XML model for linguistic paradigms, including a query language and implementation, along with a model for generating presentations and an implementation. This work provides a flexible and extensible representation for storing multidimensional linguistic paradigms; and a simple yet powerful method for accessing and analysing stored data. This model allows the easy manipulation of paradigm structure, and easy presentation of systematic patterns and variations. We believe that the XML representation will be useful for archiving linguistic paradigms and for the interchange of paradigms between programs. We also believe the presentation system supports multidimensional exploration of complex linguistic datasets, a linguistic version of what is known in the database world as online analytical processing (OLAP).

In the future, we plan to investigate the following: ordering paradigm content; generating paradigms from interlinear text; and investigating a multi-table model. Ordering the cells of a paradigm is an issue because it complicates the axes, which then require the repetition of headings. The second line of enquiry is the generation and integration of paradigm presentation into interlinear text systems, which requires a level of machine learning combined with an understanding of how to integrate different levels of linguistic description. The other issue is how to represent some of the relationships within paradigms such as the phonetic characters for a vowel and its height (eg. ϵ is a high vowel). We believe that the optimum solution for some of these problems is a multi-table model.

Acknowledgements

This paper extends earlier work by (Penton et al., 2004). This research has been supported by the National Science Foundation grant number 0317826 *Querying Linguistic Databases*.

References

Steven Bird. 1997. A lexical database tool for quantitative phonological research. In *Proceedings of the Third Meeting of the ACL Special Interest Group in Computational Phonology*, pages 33–39.

Steven Bird. 1999. Multidimensional exploration of online linguistic field data. In Pius Tamanji, Masako Hirotani, and Nancy Hall, editors, *Proceedings of the 29th Annual Meeting of the Northeast Linguistics Society*, pages 33–47. GLSA, University of Massachusetts at Amherst.

Terry Crowley. 1992. *An Introduction to Historical Linguistics, second edition*. Auckland, NZ: Oxford University Press.

Edward Finegan. 1999. *Language: its structure and use*. Fort Worth: Harcourt Brace.

John Lynch. 1998. *Pacific Languages: an Introduction*. Honolulu: University of Hawai'i Press.

Kemal Oflazer, Sergei Nirenburg, and Marjorie McShane. 2001. Bootstrapping morphological analyzers by combining human elicitation and machine learning. *Computational Linguistics*, 27(1):59–85.

David Penton, Catherine Bow, Steven Bird, and Baden Hughes. 2004. Towards a general model for linguistic paradigms.

Dan Tufis. 1989. It would be much easier if went were goed. In *Proceedings of the 4th European Conference of the Association for Computational Linguistics U.K. Manchester, 1989*.

Accepted after full paper review