

Chapter 7 Implementation

7.1 Introduction

This chapter discusses the implementation of the system. Section 7.2 provides the technical information about the system, including the system and software design decisions taken. Section 7.3 outlines the structure of the system, showing the various directories and file organisation. It reviews the directory structure of the lesson directories and explains the organisation and design of the XSL files. There are several XML file types used in the template and these are explained along with an example. This section also covers the activities, tools, audio and image components of the system. Section 7.4 provides an overview of the completed courseware. It explains the various elements of the language learning course, including the images of the relevant web pages. It discusses the printed version of the system and the CD. Section 7.5 alludes to the multilingual aspect of the system and section 7.6 provides a summary of the chapter.

7.2 Technical Information

7.2.1 System and Software Design

The field of software engineering is relatively well developed. An entire dissertation could be written about the software and development process used in this project. This section attempts to provide a concise summary of the development process, highlighting the most salient points.

The system was developed using a hybrid of the “waterfall” model and the evolutionary development model as defined by Sommerville (1996). This meant that the benefits of the structured approach offered by the waterfall model could be combined with the prototype approach used in the evolutionary development model. The waterfall model starts off with the definition of requirements (in this case the output of the development module of Hubbard’s (1996) Modular Framework discussed in chapter 6). This feeds into the system and software design phase, which establishes an overall system architecture and involves representing the software system functions in a form that can be converted into executable programs. In the implementation and unit testing phase, the software design is converted into programs and each unit (or program) is tested individually.

Integration and system testing refers to the integration of the programs as a system and the testing of the complete system. Operation and maintenance occurs when the system is installed and put into use. Maintenance involves correcting errors (and can also cover improving and enhancing the system).

In evolutionary development an initial implementation is developed, shown to the user and refined until an adequate system has been developed. Specification, development and validation are not separate activities, they happen concurrently with feedback between activities.

The software development process of this project used a combination of both models. It followed the waterfall model in so far as it started with a set of requirements as outlined in chapter 6. The overall system and software were designed based on these requirements. The system was then implemented and each program or part of the system was individually tested. The various different units were integrated

and system testing took place. The system was then handed over to the users and the necessary maintenance undertaken. Thus, at a high level, i.e. at an overall system level, the waterfall model was followed.

At a more granular level, an evolutionary development approach (Sommerville, 1996) was adopted. An outline description for a lesson was specified, but the process of specifying, developing and validating it was an iterative one. An initial version was created to enable user evaluation at an early stage in the project. (This could be considered to be an initial prototype). Intermediate versions of the system were developed, each one attempting to improve on the previous one. Lastly, the final version was developed and presented to the user.

Software design activities include the architectural design, system specification, interface design (i.e. the interface between the sub-systems, component design, data structure design (in this case, the XML file structure) and algorithm design (mainly for the XSL files). The overall architecture of the system is shown in figure 7.1 (see section 7.3 for more details).

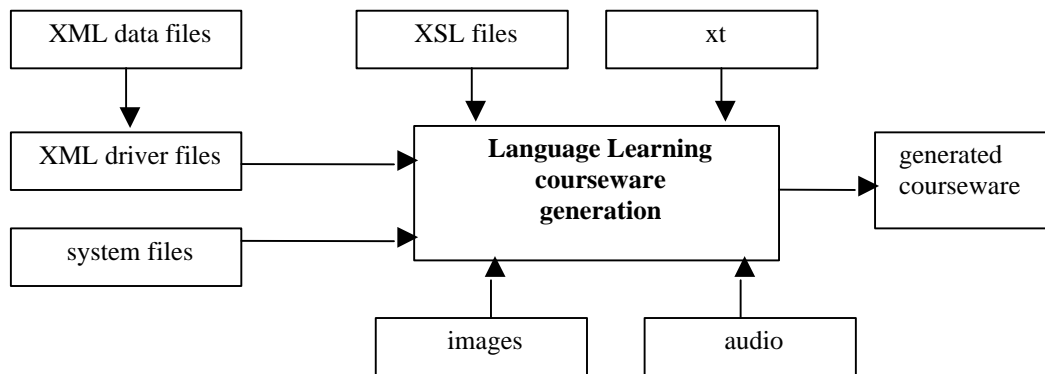


Figure 7.1 Overall system architecture

Project Management

Good software project management is essential in developing and delivery of software projects. Usually the goal is “on schedule and within budget”. Although not driven by external or commercial forces, this project had certain “intangible” deadlines, among them the delivery of a system to the endusers within a timeframe not too long after the source data was collected. Otherwise, the momentum may have been lost and the system dismissed as just another research/investigation by a foreigner and nothing more – not an uncommon feeling amongst the EL communities (Masayeva Jeanne, 1992). With software management it is often difficult to estimate the time required to develop the system, especially when new technology is being used. Thus, planning and estimating are iterative processes. Project milestones can be used to monitor the development progress of the project at certain key points.

This project differed from commercial projects (but not research projects) in that the project manager, designer, developer and tester were all the same person. While this provided tremendous insight into the strengths and weaknesses of the development “team” from the management point of view, it meant that

more self-discipline was required as there was no “external boss” to answer to. Initially, several development milestones were drawn up, including the development of a simple XML file and related XSL file for one section, one activity, one vocabulary page and one explanation page. Other milestones included bringing these XML and XSL files together to produce one lesson, and then all the lessons. There were also milestones for all the other tools. It was difficult to estimate the time required for development (especially XSL development, the processing engine or “logic” part), as it was a completely new technology to the developer (and indeed to the software development community in general). However, as the project progressed and the learning curve traversed, it became easier to estimate the time required (which diminished progressively as the project developed).

Design Quality

One important issue in this project was design quality. Many factors contribute to the quality of the design, including cohesion, coupling, understandability and adaptability. Cohesion is a measure of how closely the parts of a component relate to each other – it is the internal relationship between the parts of a component. It is desirable to have close cohesion as it means that a unit represents a single part of the problem solution. Coupling is a measure of the strength of component interconnections – it is the external relationship between components. It is desirable to have weak coupling and it means that components are (almost) independent of each other. Understandability is important because in order to be able to change the design it must first be understood. Factors that affect understandability include cohesion and coupling, naming conventions, documentation and complexity. Adaptability refers to how easy it is to change the design.

This project endeavoured to have strong cohesion and weak coupling, by having all the related elements together in their own XML file and XSL file respectively (e.g. section information). Thus, section information (and only the related elements) could be found in one file (XML for data, XSL for logic) and any changes required could be effected on the section component alone, causing no impact on the other components.

Component names (e.g. L01 ... L12 for lessons 1 to 12, XSL for XSL files) and file names (e.g. section.xml) aim to be clear. Complexity, as understood here, is an intuitive concept. Given the complexity of working with a new technology (XML technologies, especially XSL), it was decided the XSL files would be written in as straightforward a way as possible, even if this meant not using the latest “expert level” way of doing something or writing it in a more concise way.

Program Reliability

Program reliability is important in a program such as this where the user may be inexperienced and have little or no access to technical support. In general, program reliability can be achieved by avoiding the introduction of faults and by including fault tolerant facilities in the system. Defensive programming (Sommerville, 1996) involves incorporating checks for faults and fault recovery code in the program.

Given that most of the input to the system is via user mouse clicks (and typed input only in the case of end-of-lesson exercises), the introduction of faults into the system is minimised. The user does not input

any data that is stored in the system. It is important in this instance that all the links work to maintain a high level of program reliability and (equally importantly) the user's perception of program reliability. If the user perceives the program to be unreliable, s/he will be confused when something unexpected occurs and will be unsure if s/he caused the error or whether it was a problem with the program. This will undermine the user's confidence and willingness to use the system.

Defensive programming is a holistic, preventative approach to programming that aims to anticipate errors and potential problems and cater for them in the code. In this project, it is used in the XSL files to provide default values if one of the expected parameters is not passed to it (actually, with XSL, default values must be provided if a parameter is not specified when the XSL file is used). Also, XML technologies adopt a defensive strategy when a certain data item does not exist in the XML file. It assumes that it is acceptable for it not to be there and does not complain with error messages. This is good for flexibility, as some files may need to use this item while others may not.

One important thing to consider when dealing with novice users is their lack of confidence when using the computer. Users need to be reassured that they cannot 'break' the system and in the worst case scenario, they can just exit the program and start again.

Software Reuse

The template aims to be a reusable piece of software. In order to achieve this, certain design guidelines were borne in mind. A modular approach was adopted to the development of the XSL files (the main engine of the system). Care was taken with naming conventions. Individually reusable components (e.g. the sidebar) were extracted and imported into individual modules. Where possible, parts of the system were generated. For example, at a high level the HTML file names were generated, while at a lower level, the names of the link files were also generated.

System portability is another key issue to consider. Given the fact that XML technologies have been used and that one of their goals is to be computer and operating system independent, it is hoped that the system is portable (even looking to the future). Another angle is that as the linguistic community develops data storage models for linguistic data, it is hoped that the data of the system can be reused or exported into a new format (LEW 2000; LEW 2001). For example, one of the aims of LEW 2001 (LEW, 2001) was to explore applications of current database research and how they relate to the problems of linguistic databases. XML (and its related technologies) is an active field of research. It is hoped that by using XML technologies as the underlying technology for the project that the linguistic data can be integrated or converted into future standard linguistic schemas.

7.2.2 Design Decisions

Why a Web Page?

A web page lends itself to the language learning process. It allows the incorporation of multimedia elements into a screen (or page) in an almost seamless manner. Thus, images and audio parts of language lessons can be easily made available to the learners. It allows the learner to move from one screen to

another without restrictions. It enhances the sense of learner autonomy as learners can move back and forth through the system, as they like, rather than in an order mandated by the system.

Links can be used to direct the learner to related information. For example, the learner can be directed to a dictionary entry for a word (within a piece of text). Learners tend not to use the “extra” options within a system, thus providing the information online via links may encourage the learner to review such information, that would otherwise be ignored in a more traditional setting.

Another reason for choosing a web interface is that it is a known interface. With the continued growth of the Internet, people are becoming more accustomed to the look and feel of web pages. Thus, even if the members of the EL community are not familiar with the web interface, it is hoped that their potential instructors may have some prior web knowledge. As the web interface becomes more ubiquitous, it is hoped that the learning curve for the language learning system will not be too steep and that the cognitive load placed on the learner will not be too great.

By using a web interface, it is a simple process to put the pages on the Internet. This brings with it some benefits. Firstly, the language lessons are available to anyone that has access to the Internet. For EL communities that are geographically dispersed (for example, in North America), this can help to overcome the problems of distance (Buszard-Welschard, 2001). It may not be possible or feasible to provide the language learning software to all interested parties but making it available to anyone with an Internet connection can help mitigate this problem.

Secondly, the Internet allows for updates to the language learning system. Errors can be corrected and new material can be provided. Thirdly, communications between learners can potentially be established via email, discussion groups and bulletin boards. This could encourage a sense of community amongst learners, which can enhance the language learning process.

A hoped-for outcome of placing the language learning system on the Internet is to raise the profile and prestige of the EL. Often the EL is rejected by the youth of the EL community, as “something that doesn’t belong in the modern world”. By demonstrating that their language can exist on the Internet alongside English, Spanish or any other language, hopefully they will realise that it can exist in the present (and future) and not only in the past. For many, the “wow” factor has disappeared from the Internet, but in EL communities, this may still be an important factor, especially where there is limited access to computers and Internet technology. Perhaps some of this momentum could be channelled into an interest in the language.

XML Technologies

XML (eXtensible Markup Language) technologies (XML, 2000) are the technical engine of the present project. HTML is the current language of the Internet. It was initially designed to enable simple information to be shown on the Internet. However, its limitations become more apparent with the growth and complexity of the Internet. It was decided to use XML technologies in this project and this section outlines the reasons behind that decision.

The underlying principle of XML and its related technologies (henceforth referred to as XML technologies) is the separation of content (data) from its presentation (processing). Given content in a specified format, the idea is that the data can be converted into a different format. For example, XML technologies can be used to convert data from one database to another (thus facilitating data exchange) or to convert XML files to pdf files. The content provider can concentrate on the raw data, without having to worry about how it gets converted. This section outlines the main advantages of XML technologies, while its drawbacks are considered subsequently.

Benefits of XML

1. Separation of content from presentation: This allows the content providers to concentrate on what they are good at.
2. Data reuse: The same data can be "reused" and presented in many different ways. For example, dictionary information can be brief (word, meaning) or full (pronunciation, category, or audio). It also permits the conversion of data into different formats.
3. Ease of structuring data: The structuring of data is a relatively straightforward process. Rather like designing database tables, the designer can decide what information should be stored and how it should be organised. However, the type of information (e.g. numeric or character) does not have to be specified. How the data is treated (as a number or a set of characters) belongs on the "processing" side of the equation.
4. Non-proprietary software: XML does not belong to a particular company or group of individuals. It grew out of the need for a mechanism to be able to share data amongst information providers and receivers who used their own proprietary standards (e.g. different database formats) and the recognition that something more structured and flexible was required to replace HTML. It is a collaborative project, which aims to incorporate the positive features of such efforts and at the same time avoid the problems of proprietary standards.
5. Simplified SGML: SGML (Structured Generalised Mark-up Language) (SGML, 1995) is a well-defined but complex language. While theoretically sound, its complexity has prevented its widespread adoption. HTML is a (much-simplified) implementation of SGML. It has a set of pre-defined tags (e.g. for bold) that allow the user to specify how data is to be presented. It is not as theoretically sound (often ignoring missing closing tags (e.g.)). XML manages to combine the theoretical soundness of SGML, without some of its complexity. It is a simplified version of SGML, incorporating its most useful feature while omitting the more obscure or "less likely to be used" ones (XML, 1999).
6. Flexibility: HTML has a fixed set of tags. Users cannot add new ones, no matter how much they would like to. With XML, there are no such restrictions. New tags can be invented as required and the data designer can decide what they mean.

7. Unicode: XML technologies support Unicode (Unicode, 2001). Unicode is the international system developed to enable characters from all the world's languages to be displayed on the computer. With such an ambitious aim, obviously not all the character sets are currently covered (as one can imagine the coverage is less for ELs). However, a framework has been devised whereby many character sets are supported. New character sets continue to be added and the ability of XML technologies to handle these sets will continue to prove important (for example, in the teaching of minority, let alone endangered, languages).

This is by no means a comprehensive review of the advantages of XML. However, it gives an overview of the benefits that the technology offers (XML enthusiasts would no doubt add many more) and hopefully highlight its power and potential.

XML - the Downside

No technology, especially an emerging one is without its problems. XML is no exception. This section points out some of these and looks at how they may be resolved.

1. XSLT: XSLT (eXtensible Stylesheet Language Transformation) (XSL, 2001) is the language used to convert XML data from one format to another. It considers an XML file as a set of nodes in a tree and processes each node in turn. While not being extremely difficult, it does have a longish learning curve. Thus, while having nice, clearly defined XML data files is great for the data provider, the process of acting upon that data (via XSLT) may present some initial difficulties. As the XML technologies continue to mature, as tools emerge and as XSLT expertise grows, this will become less of a problem. However, in the short term, this will continue to be a real issue.

2. Evolving technology: The initial draft of XML appeared in November 1996. The first recommendation was in February 1998. The surrounding technologies have been built up since then, with the draft of XMLT being agreed in 1998. As with many new technologies, different implementations support different features or add their own (although most support the core requirements). Thus, the user has to decide which one to use and this may be a problem as the necessary expertise to choose between the various options may not be available.

Another issue that arises is that of "specified but not yet implemented" features. Such is the power and flexibility of the technology that it opens up many options and suggests many previously infeasible ideas (for example, the ability to provide a link to more than one destination from a link on a web page). However, some development time must be allowed for these new features - so not all the new features that appear in the specification may be available when users obtain the relevant software.

This problem is by no means limited to this technology and it is something that the passage of time can help ameliorate (but usually never completely eradicates). Indeed, this issue will nearly always exist if a given technology is to continue to progress.

3. Overwhelming number of options: Given the rate of growth and the plethora of implementations that are becoming available, it is easy to feel overwhelmed by the variety of options available. What XML editor should be used? What XSLT implementation should be used? Which is the best way to do *x*? What (if any) is going to emerge as the "standard environment"? Such is the scope of the technology, it is becoming increasingly difficult (if not impossible) for any one person to stay abreast of all the development in all the different areas. Edd Dumbill, one of the core XML gurus, said at the XML Expo in Germany that even he was finding it difficult to keep abreast of all that was happening in the XML world.

No one can realistically expect to stay on top of such a diverse and rapidly expanding field. The pragmatic solution lies in knowing where the relevant and up-to-date information is (usually online) and where to obtain (expert) advice (usually, via FAQs and discussion groups – (e.g. XML, 2000; XSL, 2001)).

7.3 System Structure

The designed template contains many different components. This section provides a succinct overview of the structure of the system. Figure 7.1 (p117) shows the overall system architecture. The XML data files contain the text of the language lessons (see section 7.3.4 and Figure 7.5). The XML driver files contain (administration) information about the contents of each lesson (see Figure 7.6). The system files are made up of Perl files and batch files. These files are used to automatically generate the driver and generate files. (There are also files to semi-automate the creation of the dictionary files). The XSL files are the processing engine of the template. They operate on the XML data files and produce the HTML files – see section 7.3.3 and Figure 7.4 for more details. The images and audio files are part of the courseware of the system. *xt* (Clark, 1999) is an executable program that is to carry out the conversion of XML data files into HTML files. The generated courseware refers to the HTML online web pages, the CD and also the printed version of the language learning system. The reader is referred to Appendix L, p295 (the Technical Manual) for more technical details on the system.

7.3.1 Overall Structure

At the highest level, the system contains:

- a source directory (with the source files and development infrastructure),
- Nawat directory (which contains the relevant courseware - i.e. all the files necessary to run the system),
- nawat.html (the home page which enables the users to start the course).

The system is structured as follows:

- there are twelve lessons,
- a directory each for the alphabet, the dictionary and revision,
- a general directory which contains all the text information pages (e.g. the introduction) as well as shared data such as a label file (which contains text for commonly used phrases such as “Go to top of page”),
- a directory with all the sound files,

- an image directory for all the culturally-neutral images,
- a scan directory which contains all the culturally specific scanned drawings,
- an XSL directory which contains all the XSL files,
- there is one tools directory which contains various Perl scripts (used for automatic generation of system files) and other utilities,
- a documentation directory which contains documents about the courseware.

7.3.2 Lesson Directory Structure

The lesson directory structure is as follows:

- a driver directory (which contain all the driver files),
- a generate directory (which contains all the script files),
- a HTML directory (which contains the generated HTML files),
- a source directory (which contains the XML source files).

There is a driver file for each of the XML file types. These files (which are automatically generated by a Perl script) act as wrappers around the XML data files, ensuring a complete separation between course data and 'other' data (e.g. administration items, labels for different parts of the system and the switches between English and Spanish). Section 7.3.5 contains details of the driver files.

xt (Clark, 1999) is used to generate the HTML pages. There are script files (batch files) which use the driver files to create the required web pages. Each script file specifies the driver file to be used, the XSL file which contains the conversion logic and a series of flags which specify the section (if required), the language (either English or Spanish) and display type. Section 7.3.6 contains details of the generated files.

To create the web pages for a particular lesson, a script in the generate directory (createLesson) is executed. This generates and places the generated HTML files in the HTML directory. If only one type of HTML file is required to be generated (e.g. section files), there is an individual script for that file type.

7.3.3 XSL Files

There is an XSL file for each different page type. There are only a few "main" files, while the majority of XSL files can be considered to be components that are included as required (e.g. the sidebar component). Figure 7.2 lists the XSL files and their function.

XSL is a language for expressing stylesheets. It is made up of three parts:

1. XSL transformations (XSLT) for transforming XML documents.
2. XML Path language (XPath), an expression language used by XSLT to access or refer to parts of an XML document. (XPath is also used by the XML linking specification).
3. XSL Formatting Objects (mainly for printing). It is an XML vocabulary for specifying formatting semantics.

An XSL stylesheet specifies the presentation of a class of XML documents by describing how an instance of the class is transformed into an XML document that uses the formatting vocabulary. More information is available at XSL (2001).

In this project, each XSL is used to convert an XML file into a HTML page. Sometimes several XSL files are used to create a given HTML page. While each XSL file has its own logic, the same basic logic is used throughout.

File	Function
activity_match	Logic for the matching activity
activity_matchp	Logic for the matching activity (printed version)
activity_select	Logic for the selection activity
activity_selectp	Logic for the selection activity (printed version)
activity_write	Logic for the end-of-lesson activity
alphabet	Logic for the alphabet page
dict	Logic for the dictionary pages
grammar	Logic for the explanation
grammarp	Logic for the explanation (printed version)
help	Logic for the help page
index	
intro	Logic for the text based pages
layout	
lesson	Logic for the main lesson page
lessonx	"main" file for section, explanation and vocabulary pages
page-html	Logic for the section icons
print	Logic for the print version
revision	Logic for the revision page
section	Logic for the section page
sidebar	Logic for the sidebar element
tip	Logic for the tip page
vocab	Logic for the vocabulary

Figure 7.2 List of XSL files and their function

Each XSL file is made up of templates (somewhat akin to functions/procedures/subroutines in procedural programming or methods in object oriented programming). Each XSL template deals with different nodes in the XML source file. Figure 7.3 shows the pseudocode for the section XSL file.

Figure 7.4 shows sample code from the section.xsl file – it shows the code for the conversation node template (i.e. the code that processes the “conversation” nodes in the xml data file). Nodes from the XML file have been highlighted in bold. Each line of the conversation (e.g. <p1> or <p2>) is displayed on one line and a link is given to its corresponding the audio file (stored in the { \$sounddir } directory). Each conversation line shows who is speaking (<who>) and what s/he is saying (<what>). If a translation is being shown (i.e. if the translation flag, \$tr, is true), then the translation is shown in the required language, which is either English (<english>) or Spanish (<spanish>). Further details are available in the Technical Manual.

```

Lesson Page

Print sidebar
Print title (in either English or Spanish)
Print lesson info (lesson number, title and meaning)
Do lesson body
    For each element
        For each section
            Compose image and link variables
            Print image, link, title
            Print meaning in either English or Spanish

        For each item (exercise, explanation, vocabulary)
            Compose image and link variables
            Print image, link, title

```

Figure 7.3 pseudocode for the main lesson page

```

<xsl:for-each select="p1|p2">
<tr>
    <td>
        <a href="{ $sounddir}/{ $nlid}/{ $aid}">
        
        </a>
    </td>
    <td>
        <xsl:value-of select="who"/>
        <xsl:text>: </XSL:text>
    </td>
    <td>
        <xsl:value-of select="what"/>
        <br/>
        <!--print translation if required -->
        <xsl:if test="$tr='true'">
            <font color="green">
                <xsl:choose>
                    <xsl:when test='$lang="english"'>
                        <xsl:value-of select="english"/>
                    </xsl:when>
                    <xsl:when test='$lang="spanish"'>
                        <xsl:value-of select="spanish"/>
                    </xsl:when>
                </xsl:choose>
            </font>
            <br/>
        </xsl:if>
    </td>
</tr>
</xsl:for-each>

```

Figure 7.4 Sample code from section.xsl

All the XSL files can be found in the XSL directory on the CD.

7.3.4 XML Files

XML – an Introduction

The web site www.w3.org/XML has a comprehensive introduction to and explanation of XML. A brief summary is provided here.

XML development started in 1996 and has been a W3C (World Wide Web Consortium (W3C, 2000)) standard since February 1998. The eXtensible Mark-up Language (XML) is the universal format for structured documents and data on the Web. It is a method for putting structured data in a text file. XML is a set of rules or conventions for designing text formats for data. It produces files that are easy to generate and read (by a computer) that are unambiguous and that avoid common pitfalls (e.g. such as lack of extensibility, lack of support for internationalisation/localisation, and platform-dependency). XML looks a little bit like HTML but is actually quite different. XML uses tags only to delimit pieces of data and leaves the interpretation of the data completely to the application that reads it

The following is taken from Extensible Mark-up Language (XML) 1.0 (Second Edition) W3C Recommendation 6 October 2000 (XML, 2000):

“XML documents are made up of storage units called entities, which contain either parsed or unparsed data. Parsed data is made up of characters, some of which form character data, and some of which form markup. Markup encodes a description of the document's storage layout and logical structure. XML provides a mechanism to impose constraints on the storage layout and logical structure.”

The design goals for XML are:

1. XML shall be straightforwardly usable over the Internet.
2. XML shall support a wide variety of applications.
3. XML shall be compatible with SGML.
4. It shall be easy to write programs which process XML documents.
5. The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
6. XML documents should be human-legible and reasonably clear.
7. The XML design should be prepared quickly.
8. The design of XML shall be formal and concise.
9. XML documents shall be easy to create.
10. Terseness in XML markup is of minimal importance.

What is XML?

XML is a set of guidelines for designing text formats for data that is easy to produce and read, that are unambiguous, extendable and platform independent. XML is a method for putting structured data into a text file. XML looks a little like HTML but, unlike HTML, only defines data, not the format of how that data is to be presented. Figure 7.5 shows an example of how XML files and XSL files are combined to produce HTML pages.

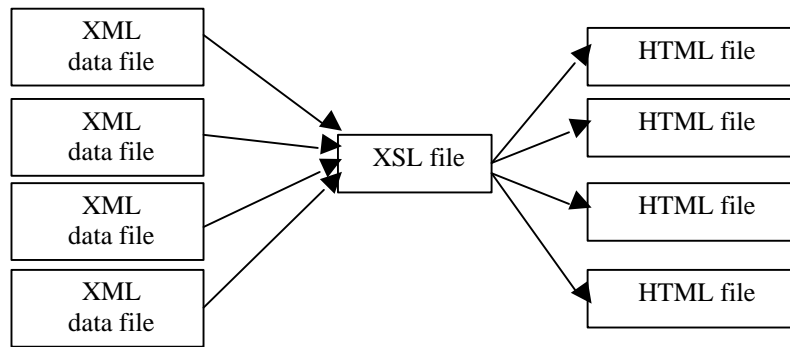


Figure 7.5 How XML files and XSL files are combined to produce HTML files

XML allows data to be organised in whatever way one chooses. By contrast, HTML comes with a fixed, pre-defined set of tags (for example, <body> or <p> for paragraph). With XML, it is possible to define tags that can be used to specify the meaning of data. Let us illustrate XML with an example. Suppose we wanted to design a mini-dictionary. The entry for a given word would contain the pronunciation, a gloss, the meaning and an example. Each data item is identified by a start and end tag (see Figure 7.6).

```

<entry>
  <word>dog</word>
  <pronunciation>dog</pronunciation>
  <meaning>A highly variable domestic mammal(Canis familiaris)
</meaning>
</entry>
  
```

Figure 7.6 Sample XML file for a dictionary

Later in the development of a CALL application, we might decide to add a category (hypernym) element (for example, “dog” can belong to the “mammal” category). To do this we could simply add an additional tag to our word entry (see Figure 7.7).

```

<entry>
  <word>dog</word>
  <pronunciation>dog</pronunciation>
  <meaning>A highly variable domestic mammal(Canis familiaris)
  <category>mammal</category>
</entry>
  
```

Figure 7.7 Sample XML file for a dictionary with additional tag

To display the dictionary information on a web page, we need to write an XSL file to process the information and specify its format. Figure 7.8 shows some sample XSL code for a dictionary. The tags from the XML data file are highlight in bold. This section of code prints the word (<word>) and the pronunciation (<pronunciation>) for each entry (<entry>) in the dictionary.

```

<xsl:template match="dict">
  <h2> Basic dictionary</h2>

  <-- for each entry in the dictionary -->
  <-- print the word and its pronunciation -->

  <xsl:for-each select="entry">
    <table>
      <tr>
        <td>
          <xsl:value-of select="word">
        </td>
        <td>
          <xsl:value-of select="pronunciation">
        </td>
      </tr>
    </table>
  </xsl:for-each>
</xsl:template>

```

Figure 7.8 Sample XSL code for a dictionary

XML Sample

The above example was based on a simplified XML file for a dictionary. In the template, each web page type has a different XML structure (e.g. there are XML files for section, vocabulary and activity data). Figure 7.9a shows part of the XML file for the section data file and Figure 7.9b shows an example (see the CD included with the thesis for the other XML data files). It contains title (<title>), theme (<theme>) and text (<text>) nodes. The text node contains several contents nodes (<content>), one for each phrase (either <p1> or <p2>) in the conversation. Each content node contains the speaker information: who (the name of the speaker, <who>), what (the speaker says, <what>) and the meaning (<meaning>) in English (<english>) and Spanish (<spanish>).

XML File Structure

```

<section id = " ">
<title> ken mutukay?
</title>

<theme>
  <english>    </english>
  <spanish>    </spanish>
</theme>
<text>
<content>
  <p1>
    <who> </who>
    <what> </what>
    <english> </english>
    <spanish> </spanish>
  </p1>
</content>
</text>

```

Figure 7.9a Sample blank XML data file for the section data file

Example

```
<section id ="2">
<title> ken mutukay?
</title>

<theme>
  <english>What's your name?
  </english>
  <spanish>&#191;c&#243;mo se llama?
  </spanish>
</theme>
<text>
<content>

      .....

  <p1>
  <who>Genaro</who>
  <what>naha nutukay Genaro.</what>
  <english>My name is Genaro.</english>
  <spanish>Me llamo Genaro.</spanish>
  </p1>
      .....

</content>
</text>
```

Figure 7.9b Sample XML data file for the section data file

Note that the section <p1> ... </p1> can be repeated for all the phrases in the conversation. Note also that special characters are used for those Spanish characters that are not used in English. Appendix D, p186 provides a list of all the relevant (Unicode) characters.

7.3.5 Driver Files

The driver files permit the separation of "pure" XML data from the "administrative" XML data. The driver file for a section is presented here in Figure 7.10. The first four lines are general XML administration lines. Then follows a list of the files that are referred to later on in the file. Note that the code in Figure 7.10 shows the driver file for the section pages of Lesson 1. The common elements used in the file are the sidebar.xml, labels.xml, tip.xml, topbar.xml and bottombar.xml files. These are common to all lesson files and it makes easier to effect global changes (to the layout) as amendments only need to be made in one place and will filter through once the lesson pages are regenerated. Figure 7.10 shows the three section files for Lesson 1, namely: l01s1.xml, l01s2.xml and l01s3.xml. More information on the driver files can be found in the Technical Manual.

Note that the file does not contain any "pure" data. Using this driver file, section pages can be dynamically generated for each section in both English and Spanish. Note also that there are defaults for the flags used by this driver file. For example, Figure 7.11 show a line from the section.xsl file with the default value for the tr (translation) flag.

```

<?XML version="1.0"?>
<?XML-stylesheet href="../../general/lessonx.xsl" type="text/xsl"?>
<?cocoon-process type="xslt"?>
<?cocoon-format type="text/plain"?>
<!DOCTYPE page [
    <!ENTITY sidebar SYSTEM "../../general/sidebar.xml">
    <!ENTITY labels SYSTEM "../../general/labels.xml">
    <!ENTITY tip SYSTEM "../../general/tip.xml">
    <!ENTITY topbar SYSTEM "../../general/topbar.xml">
    <!ENTITY bottombar SYSTEM "../../general/bottombar.xml">
    <!ENTITY title SYSTEM "../source/title1.xml">
    <!ENTITY section1 SYSTEM "../source/l01s1.xml ">
    <!ENTITY section2 SYSTEM "../source/l01s2.xml">
    <!ENTITY section3 SYSTEM "../source/l01s3.xml">
]>

<lesson id="1" tr="false" lang="english" langnum="0" type="section">
    &sidebar;
    &labels;

    <unit uid="1">
        &title;
        &tip;
        &topbar;
        &section1;
    </unit>

    <unit uid="2">
        &title;
        &tip;
        &topbar;
        &section2;
    </unit>

    <unit uid="3">
        &title;
        &tip;
        &topbar;
        &section3;
    </unit>
</lesson>

```

Figure 7.10 Section driver file

```
<xsl:param name="tr">false</xsl:param>
```

Figure 7.11 Specifying the default value (false) for the tr (translation) flag

7.3.6 Generate Files

The generate files are used to generate the HTML pages. Figure 7.12 lists the generate files found in the lesson directories and their function. (The other directories - alphabet, dictionary, revision and general - have fewer generate files, but their function is analogous). Figure 7.13 shows the generate file for the explanation files. It has a line for each file that is must generate. It specifies the name of the input XML file, the XSL file that will process the XML data, various parameter settings (e.g. language and section number) and the name of the output file. More information is available in the Technical Manual (see

File	Function
activityMatch.bat	Creates the matching activity page for section 3
activitySelect.bat	Creates the select activity pages for sections 1 and 2
createLesson	Creates all the pages of the online lesson
exercise	Create the end-of-lesson exercise page
grammar	Creates the individual grammar pages for each section
grammarAll	Creates the grammar page for the lesson
print	Creates the print version of each section
section	Creates the section pages
vocab	Creates the individual vocabulary pages for each section
vocaball	Creates the vocabulary page for the lesson

Figure 7.12 List of files in the generate directory

```

..\..\tools\bin\xt      ../drivers/l01g.xml      ../../XSL/lessonx.xsl
lang="english" langnum="0" ext="html" sid="1" ../HTML/l01g1a_eng.html
..\..\tools\bin\xt      ../drivers/l01g.xml      ../../XSL/lessonx.xsl
lang="spanish" langnum="1" ext="html" sid="1" ../HTML/l01g1a_esp.html

..\..\tools\bin\xt      ../drivers/l01g.xml      ../../XSL/lessonx.xsl
lang="english" langnum="0" ext="html" sid="2" ../HTML/l01g2a_eng.html
..\..\tools\bin\xt      ../drivers/l01g.xml      ../../XSL/lessonx.xsl
lang="spanish" langnum="1" ext="html" sid="2" ../HTML/l01g2a_esp.html

..\..\tools\bin\xt      ../drivers/l01g.xml      ../../XSL/lessonx.xsl
lang="english" langnum="0" ext="html" sid="3" ../HTML/l01g3a_eng.html
..\..\tools\bin\xt      ../drivers/l01g.xml      ../../XSL/lessonx.xsl
lang="spanish" langnum="1" ext="html" sid="3" ../HTML/l01g3a_esp.html

```

Figure 7.13 The generate file for the explanation files

7.3.7 Tools

The tools directory contains the script files used to automatically generate the driver and generate files.

Figure 7.14 shows the relevant files.

File	Function
bin	Directory which contains the xt executable and perl software
prepareBatch	Prepares all the generate file
prepareDrivers	Prepares all the driver files

Figure 7.14 Contents of the tools directory

7.3.8 Activities and Exercises

The activities and exercises used in the template are based on Hot Potatoes from Half-Baked software (Hot Potatoes, 2001). Hot Potatoes is an Authoring Tool specifically for CALL exercises. While the use of an Authoring Tool may seem contrary to the aim of the project, this is not the case. Hot Potatoes is not, and does not claim to be, a complete CALL Authoring Tool - it focuses on the creation of interactive CALL exercises. In keeping with the ethos of the project to avoid reinventing the wheel where possible, Hot Potatoes software was used as the basis of the interactive exercises.

Hot Potatoes is aimed at the language teacher CALL author and provides an easy-to-use and friendly user interface. Normally with Hot Potatoes, authors can create six different types of exercises online by running an executable which generates the required web page. While this interface is suitable for creating individual exercises (where just a few web pages are required), it was not appropriate in the context of the template. As a total of 96 (4 per lesson x 12 lessons x 2 languages (Spanish and English)) activity pages were required, it was not considered feasible to use Hot Potatoes in interactive mode. Each modification would have required reloading the Hot Potatoes software to make the change, which would have been time consuming and not suited to automatic regeneration and update of courseware. Furthermore, it was important to ensure that the Spanish and English versions of the courseware stayed in synchronisation and this would have been difficult to control in the normal Hot Potatoes usage scenario.

It is important to be able to generate exercises automatically using the data from XML files. In order to be able to generate the exercises in this manner, selected parts of the Hot Potatoes software were adapted and integrated into the template system design (see Appendix I, p195). The resulting design was that two XML components are used to produce the web pages. One XML file with "pure" data with just the questions and answers and another XML file with the JavaScript code used in the activity. This allowed changes to be easily made to the questions and answers and allowed the tight coupling between the Spanish and English versions to be maintained. This means that once the XML data files are populated or updated (during course development), the entire courseware can be regenerated at the press of a button. It also ensured that the template was reusable, as the language data for the exercises was completely separate from the JavaScript code used to produce the pages and could therefore be replaced with data for a different language in the future. Figure 7.15 shows how the Hot Potatoes software was integrated into the template.

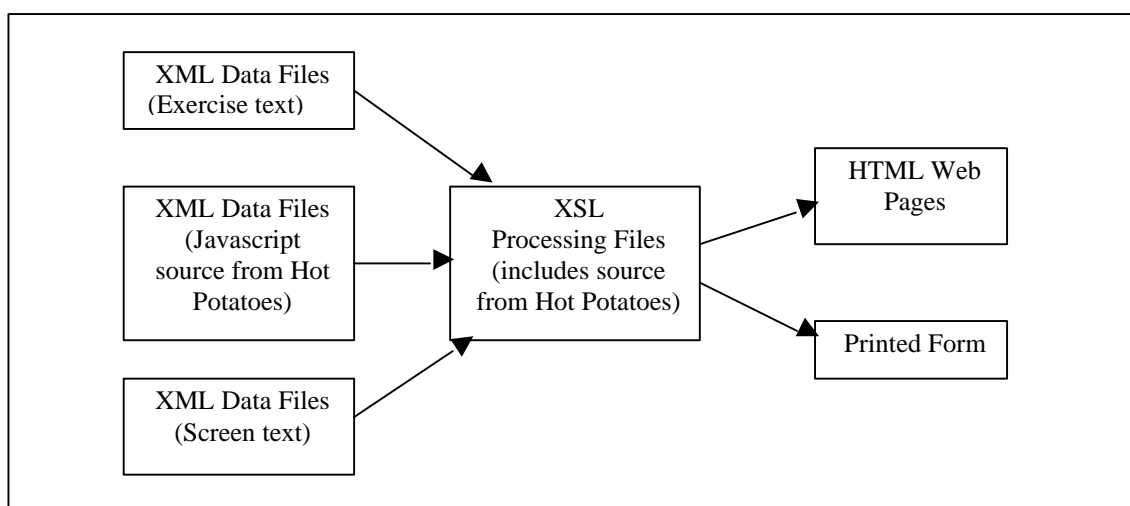


Figure 7.15 Integration of Hot Potatoes software into the template

7.3.9 Audio

The conversations were recorded on a mini-disc recorder (Sony MZ-R70). It was very lightweight and portable (ideal for fieldwork). The microphone was a Sennheiser e8155, a relatively heavy but high quality microphone. The files were saved on the mini-disc and subsequently converted into .wav format and saved in files on the computer. The .wav files were then broken down into individual lessons and

sections. The files were edited using SoundForge (version 4.0). Coughs, gaps and other undesirable content were removed. Some minor adjustments to volume and speed were made in a few areas. The files were converted to mp3 format (as they were 10 times smaller and of very similar audio quality).

Once each conversation had been tidied up, each phrase was extracted and saved in a separate file. The other option of noting the start and end time of each phrase was deemed more cumbersome and harder to incorporate into a web environment. Subsequently, some phrases were modified (usually a few words deleted) for use in the activities and exercises.

Finally, individual words were extracted and saved as xxx.mp3 (where xxx is a given word) for use in the dictionary. It would have been better to have recorded each word separately, but this was not possible.

7.3.10 Images

Images play an important part in the learning process. One criticism that emerges vis-à-vis generic templates for CALL programs is that they are not culturally specific (either in terms of cultural information or images). Thus, it was important to have culturally relevant images. However, as the aim was to create a culturally neutral template, images from a given culture could not be used in the generic parts of the system.

It was decided that culturally specific images would be used for each section and generic images in the activities and exercises. The generic images principally came from a database of images from the University of Victoria, Canada (UVIC, 2001). The culturally specific images were drawn by Jhony Cruz Ventura, a Salvadorian during my first field visit (December 2000 – January 2001).

Some of the alleged "culturally neutral" images are not completely culturally neutral. For example, an image of someone waving as a form of greeting would not be suitable in many parts of Asia. A more appropriate image could of course be substituted. It is a difficult area to get right. For example, a database of culturally neutral images had a dictionary with Japanese writing on it (Hatasa, 2001). Postman Pat, a children's cartoon character in the United Kingdom is culturally unacceptable in Japan as he only has 4 fingers (a sign for the Japanese that he is a member of the Japanese mafia, the Yakuza).

7.4 Completed Courseware

This section provides a summary of each element of the completed courseware. It covers the components of the language learning course (i.e. the online version of the system), the printed version (the off-line version) and the CD. Where appropriate, images of the relevant web pages are shown. See Appendix J, p196 for the printed version of the courseware.

7.4.1 Language Learning Course

7.4.1.1 Introduction

The introduction page provides a brief introduction to the system. It outlines the purpose of the system and the various components. A bullet point format is used for the presentation of information, as people who read the web tend not to like having to read long segments of text online.

7.4.1.2 Language Learning Tips

There are many strategies for learning a language. Most learners who do use such strategies are usually unaware that they are using a particular strategy. Many learners are unaware that such strategies exist. This component lists various language learning strategies (adapted from Oxford, 1991, 1996b). The format selected is that of bullet points and the language chosen is informal. It is not expected that every learner will diligently read this part of the system and apply all the listed strategies. However, some learners may find something that appeals to them and that they can use as part of the learning process.

7.4.1.3 Course Tutorial

This course is designed as a standalone course. Therefore, it is imperative that it contains a "how to use" tutorial. The tutorial explains how to use the system, with a bullet point explanation of each element and an image of the relevant screen.

7.4.1.4 Language Lesson

The language lesson is the most important part of the system. There are twelve lessons in the system and each lesson has the same format. Each lesson contains three sections, a combined explanation of each section, a lesson vocabulary and an end-of-lesson exercise. There is a main lesson page that contains links to the three sections within the lesson, the explanation and vocabulary summaries as well as a link to the end-of-lesson exercise. Figure 7.16 shows the web page for Lesson 1. Each link has an associated image or icon. In the case of the sections, the associated culturally specific image is shown, along with the theme of the section in Nawat and either a Spanish or English translation. Note the shaded sidebar. This appears on most of the web pages and enables the learner to move around the system (and get back to somewhere "safe" if required).

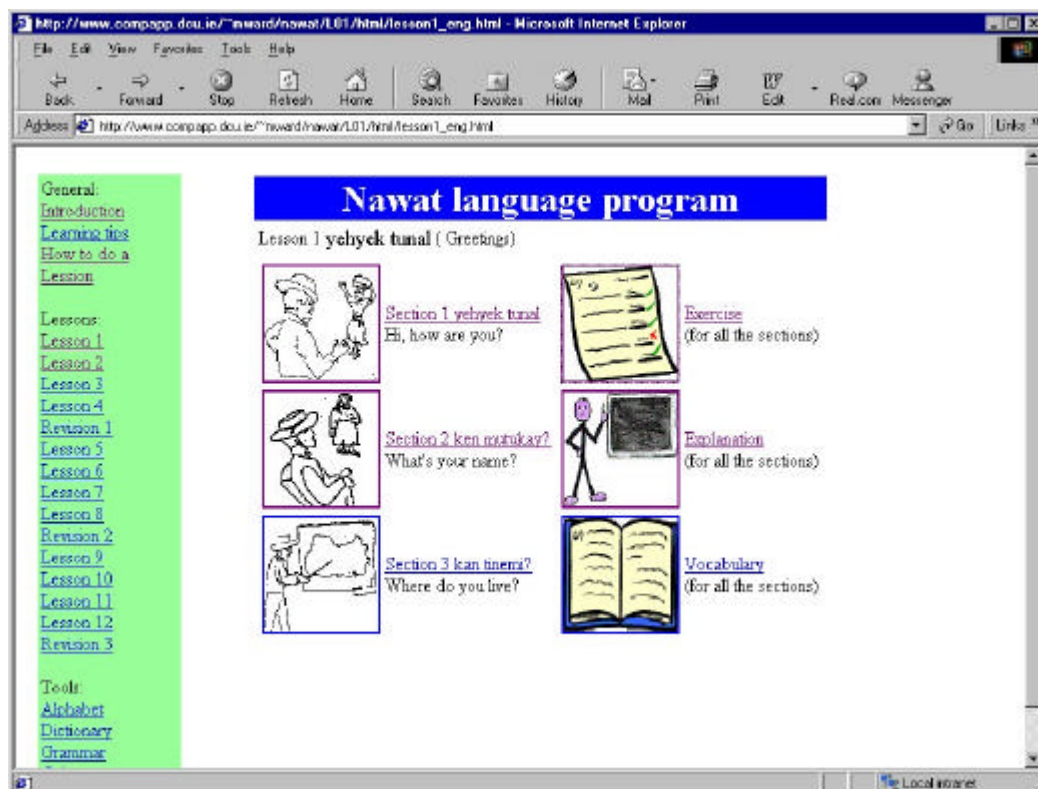


Figure 7.16 Lesson page

7.4.1.4.1 Section

The section element is the core part of the language lesson. Figure 7.17 shows an example of a section page. It contains the conversation in text format with a link to an audio version. There is also an option to view a translation of the conversation (either in English or Spanish). There is also an audio link for each phrase of the conversation, so that the learner can focus on that phrase alone. The sidebar link on the left-hand side contains a link to all the language lessons, revisions and other utilities provided by the system. Thus, the learner can move freely through the system. There is a row of icons at the top of the page with links to the other elements of the current section. These are: section (to return to the conversation page); activity (to go to the activity page); vocabulary (to go to the vocabulary page); and help (to go to the help page). The learner can return to the main lesson page by clicking on the lesson number or title.

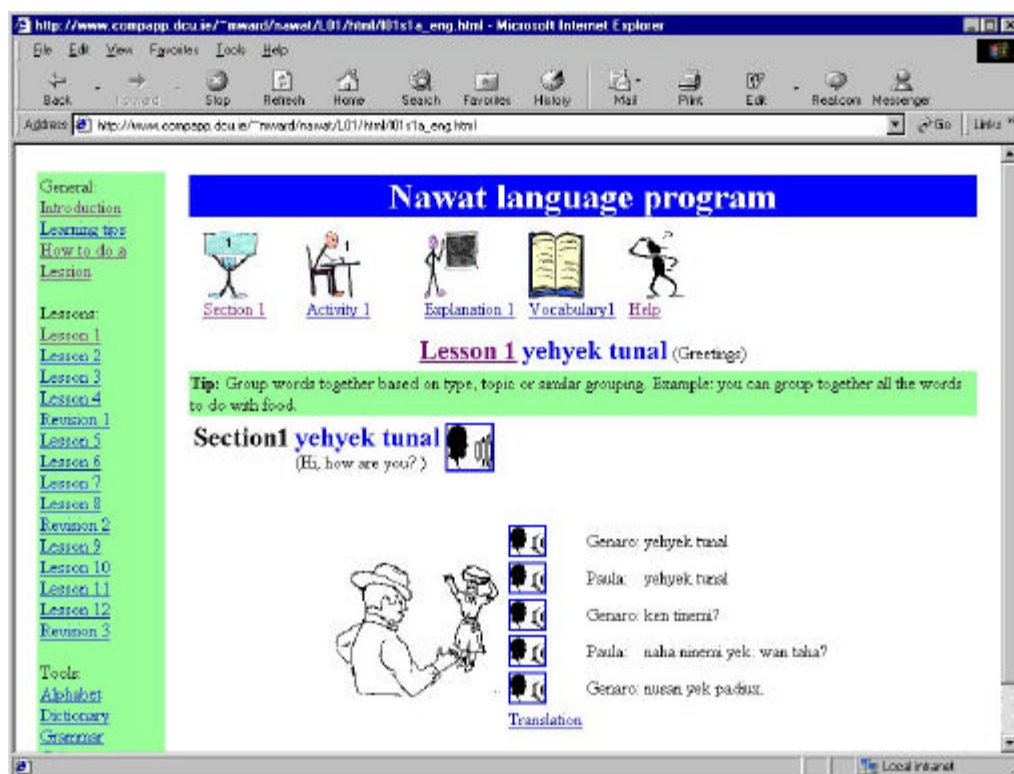
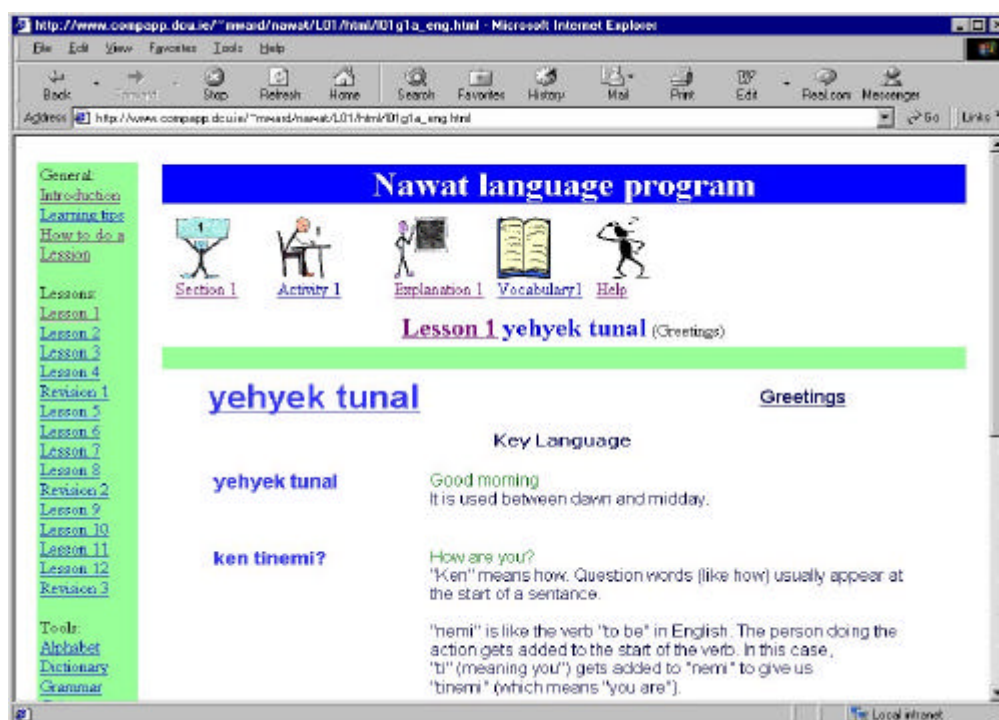


Figure 7.17 Section Page

7.4.1.4.2 Explanation

The explanation page for each section contains the principal phrases used in the section, along with a translation and an explanation. The language used is simple and non-technical. The sidebar with links to the main elements of the course is on the left-hand side. The section icons at the top of the page provide links to other elements of the section. Figure 7.18 shows an example of an explanation page.



7.18 Explanation page

7.4.1.4.3 Vocabulary

There is a vocabulary page for each section that shows the new words used in that section. It contains the sidebar on the left-hand side as well as the icons that link to the other components of the lesson at the top of the page. An example of a vocabulary page is shown in Figure 7.19.

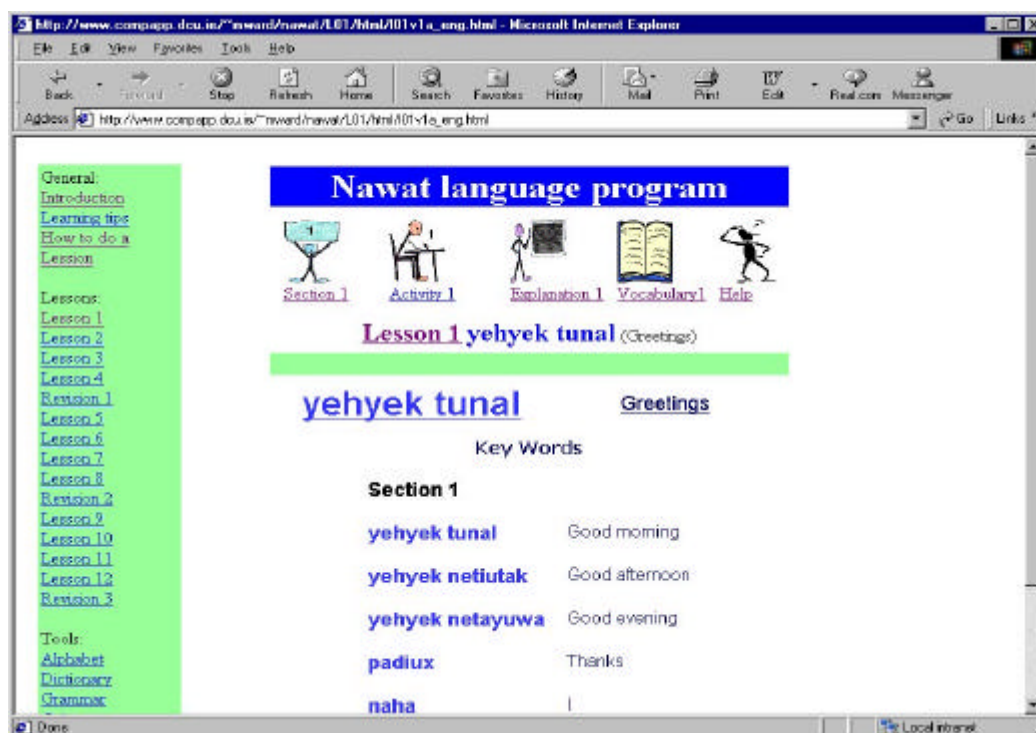


Figure 7.19 Vocabulary page

7.4.1.4.4 Activity (Type 1)

Activity Type 1 is used in sections 1 and 2 of each lesson. It contains a combination of questions. These include:

- listen to the audio file and select the item which matches what the speaker says,
- look at the image file for the question and answer the question relating to it,
- translation questions from Nawat to either English or Spanish or vice versa.

There may be more than one correct answer. There is no limit to the number of times a learner can attempt the activities. Each time the page is loaded, the questions and their corresponding answers appear in a random order. This is to avoid the situation where students attempt to get all the questions right by remembering the correct answers from previous attempts. Figure 7.20 shows an example of an activity (type 1) page.

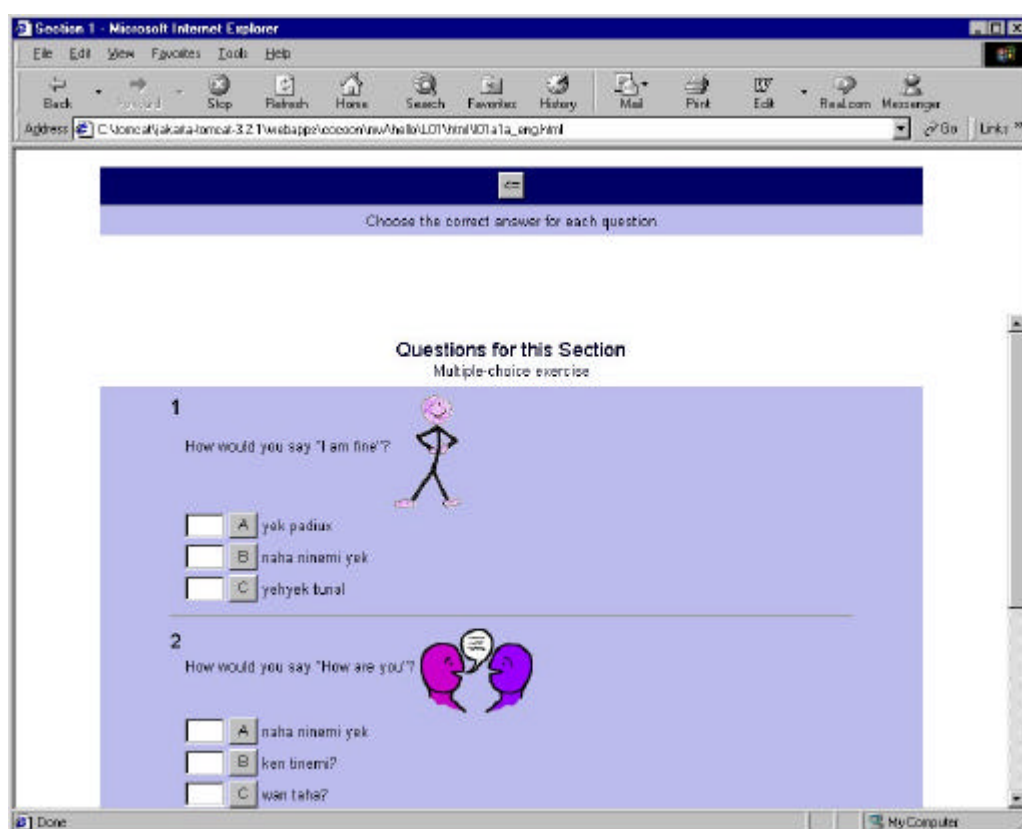


Figure 7.20 Activity (Type 1) page

7.4.1.4.5 Activity (Type 2)

Activity Type 2 is used in section 3 of each lesson. It involves matching pairs of items, which may be words or phrases. The learner has to move an item from the list on the right hand side to its matching item on the left-hand side. There is only one correct answer. If the learner matches an item incorrectly, after the "test" button has been pressed, it is returned to the right hand side of the screen, to be rematched by the learner. An example is shown in Figure 7.21.

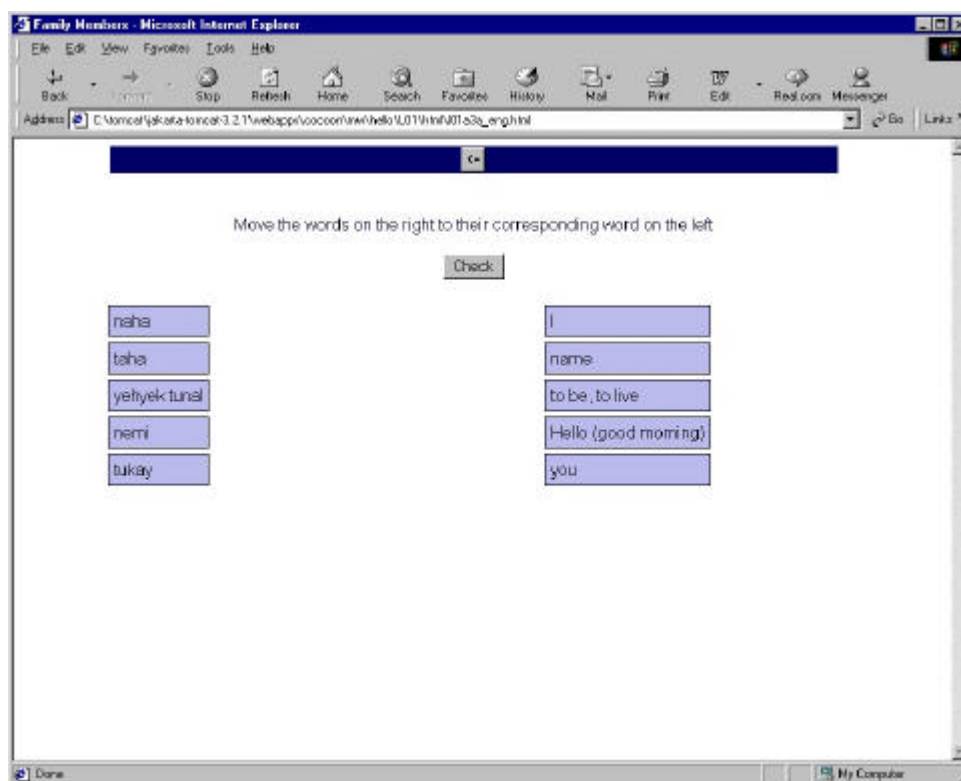


Figure 7.21 Activity (Type 2) page

7.4.1.4.6 Exercise

Each lesson has an end-of-lesson exercise. The exercise draws on elements from each of the three sections within the lesson. Learners are asked a question to which they have to type a reply (language production). However, if the learner does not know the answer, a correct response (there may be others) is provided by means of a link to an audio file with the spoken correct answer. Thus, while the learner is provided with the answer, s/he still has to understand it and be able to reproduce it. There is also a hint facility available to tell the learner what part of the typed answer is correct and it can also provide the next correct letter of the answer (see Figure 7.22).

7.4.1.5 Revision

The revision sections occur after every four lessons. They contain a review of the main grammar items covered in the preceding lessons. Each item is introduced with an explanation and several examples are given. Care has been taken to ensure that, where possible, no overtly technical language is used. This section was added based on a courseware developer request. See section 8.3, p146 for more details.

7.4.1.6 Alphabet

The alphabet page lists all the letters in the Nawat alphabet, along with a sample word and its pronunciation. If a letter differs from its usual pronunciation in the learner's native language (for example, the "h" in Nawat is like the "j" in Spanish), this is reported to the learner. See Appendix C, p185 for details of the alphabet used in the courseware.

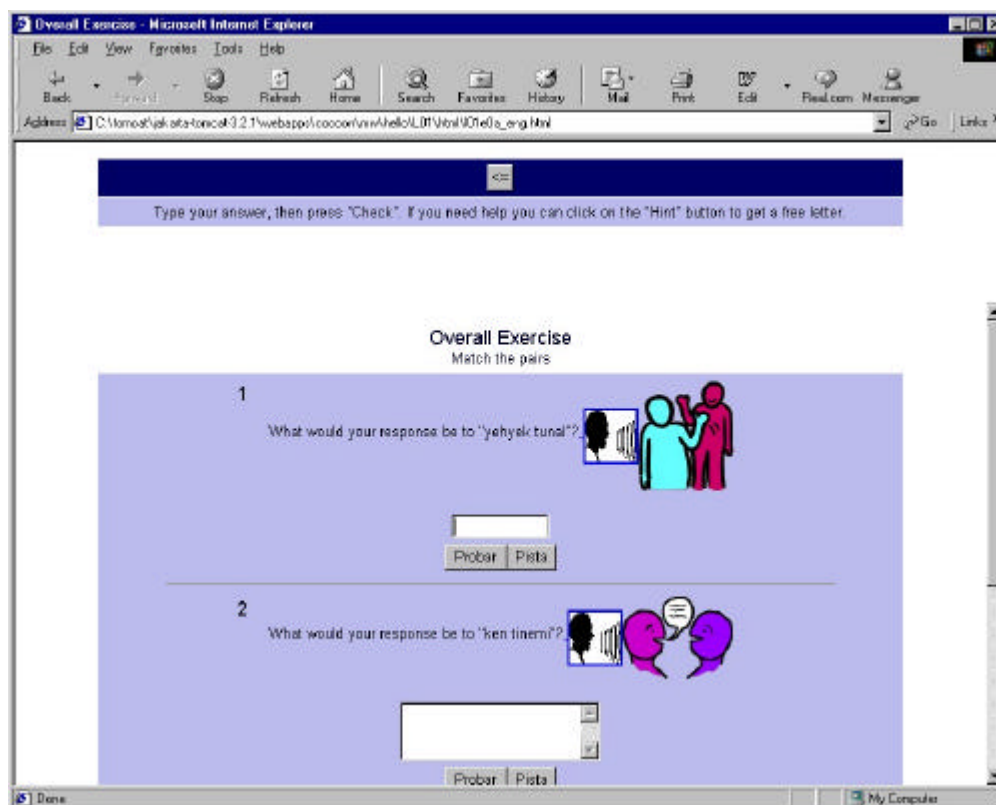


Figure 7.22 Exercise page

7.4.1.7 Dictionary

There are three types of dictionary in the course. One is just a simple list of all the Nawat words used in the course. When learners click on a word, they are brought to the full dictionary entry for that word. There is also a basic dictionary, which contains the Nawat words and their meanings. The full dictionary entry contains the Nawat word, its meaning and other information such as word type (e.g. noun or verb) and semantic category (e.g. family and food). There is an audio link that the learners can select to hear the pronunciation of the word.

There is currently a process underway within the linguistic database community to come up with some standard format(s) for storing language data (LEW, 2000; LEW 2001). One thread within this movement is that of dictionary storage. No formats have been fully defined but is hoped that by using an XML structure, at some future point the Nawat dictionary could migrate to the new format and thereby make it accessible to a wider audience.

7.4.1.8 Grammar

The grammar link points to a page that combines the three revision (essentially grammar) summaries.

7.4.1.9 Culture

Language learning and cultural knowledge are intertwined. The system contains a brief introduction to the Pipil culture. The culture section could be greatly expanded. Indeed, there are plans for people in El Salvador to work on this part of the system (see section 8.3, p146 for more details). The format of the

XML source files for the cultural section is very simple, hopefully making it easy for non-technical people to add information. One interesting feature is a song composed and sung by one of the native speakers.

7.4.2 Printed version

It was important to have a printed version of the system. Although the system was designed for online use, many more potential learners could be reached by providing a printed version of the system. The original idea was to use Formatted Objects (FO) technology (Miloslav, 2001) within the XML technologies, but this was not possible (given the steep learning curve of Formatted Objects and the project timescales). Instead, XSL files were developed to generate the printed version and the results copied into an MS word document. This is not a particularly efficient way of creating the printed version but was a pragmatic solution. Individual pages of the printed version can, of course, be printed directly from the web browser.

The printed version provides all the information that is available in the online version. There is a format change for most pages as the sidebar panel on the left-hand side is not relevant. Also, all the [Return to top] links have been removed. The format of each section is also different. Figure 7.23 shows a sample section page from the printed version. It contains the conversation, the explanation and the activity (see Appendix J, p196 for full printed version). As a dictionary is provided at the end of the book, no section specific vocabulary is provided for each section.

Lección 1 : yehyek tunal


Sección: 1

yehyek tunal

(Buen Día)

Información Importante

Genaro: yehyek tunal
Paula: yehyek tunal
Genaro: ken tinemi?
Paula: naha ninemi yek. wan taha?
Genaro: nusan yek padiux.



ken tinemi? ¿Qué tal?
"Ken" significa "cómo". Palabras para preguntas (por ejemplo cómo) normalmente vienen al inicio de la oración. "nemi" es como los verbos ser y estar. La parte para la persona que hace la acción se agrega al inicio del verbo. En este caso, se agrega "ti" (para "usted") a "nemi" que da "tinemi" (que significa "usted es/está").

naha ninemi yek. Estoy bien.
"ni" es la parte para "yo", y entonces "ninemi" significa "yo soy" o "yo estoy". Se puede omitir el "naha" (yo) porque "ninemi" ya tiene la parte de "yo". "yek" significa "bien".

Figure 7.23 Sample page from the printed version of the courseware

7.4.3 CD

The source directory, the Nawat directory (with the "release" of the system) and the nawat.html home page are included on the CD. The idea of the CD is to enable people who have access to a computer (but not the Internet) to have access to a copy of the system.

7.5 Multilingual System

The program was originally developed to teach Nawat through Spanish - Spanish being the language of the target users. However, as potential testers in Ireland were unlikely to be Spanish speakers, an English option was added. While this did increase the workload somewhat, it demonstrated that the template (which was intended to teach any language in a given language) could teach a given language in any language. By having both an English and Spanish version available online, it increases the potential range of the program. It is available as a resource for the international linguistic EL research community. Also, there may be other people interested in studying Nawat who are not Spanish speakers.

7.6 Summary

This chapter provided a description of the technical side of the project, including its software engineering base and design decisions. As with many projects, the development process was a mixture of top down and bottom up. The general outline and required structure of the system were developed in the design phase. The actual development process was bottom up in so far as the initial development of the XML and XSL files. This was necessary due to the fact that it was a new technology and the nuts and bolts had to be learnt. Along the way, once the possibilities of the technology had been better understood, modifications and enhancements were made, but no fundamental design elements were altered. In this chapter, the structure of the template was discussed, along with details of the lesson structure. Sample code was provided for the various directories. The actual courseware contents were presented, with images of the various screens shown. Information was provided on the printed version and multilingual aspect of the system.