Chapter 4
Building and displaying the clause cube using XML[59]

4.1 Introduction

The text of the Hebrew Bible is analysed from different linguistic disciplines, such as phonology, morphology, morpho-syntax, syntax, semantics, etc. It is even possible, and very helpful, to integrate these contributions using an interlinear format or table structure. A whole Bible book can, for example, be analysed clause by clause, indicating the various analyses in a collection of interlinear tables. Although this makes perfect sense for someone who studies the work in a linear fashion, it does not facilitate advanced research into linguistic structures and other phenomena. If the data could be transferred into a proper electronic database, one could create a database management system to view and manipulate the data according to the needs of linguists and exegetes.

Although the interlinear tables already resemble the tables in a relational database very closely, there is one important difference: each record or clause is represented by a unique table while records in a relational database table are similar rows in one table, all with the same structure. A typical relational database table for capturing linguistic analyses could use syntactic functions as the names of attributes or fields. Each clause could then be a row and its elements rearranged and categorised accordingly. However, one will need a large number of columns to capture all possible syntactic functions, many of which will contain null values because the structures of sentences vary significantly. Furthermore, for every language module that is added to the data store one will have to add another set of columns, aggravating the sparsity problem even further. Alternatively, one could use a parallel table linked by unique keys or references. To extract the related data one would have to use joins to collect the data from the various tables. This implementation will also

[59] This chapter is a revised version of a paper read at the Israeli Seminar on Computational Linguistics (ISCOL), Haifa, Israel, 29 June 2006 ("Building and displaying a Biblical Hebrew linguistics data cube using XML" (see Kroeze, 2006).

lead to much redundancy, since the words or phrases will have to be repeated in each table.

If one takes the word groups of the clauses as a starting point to structure the database and store data such as NP, subject, agent, etc. as attribute values, the structure problem is solved to a large extent, since each clause contains only a limited number of phrases (a maximum of five per clause in Genesis 1:1-2:3). The problem of redundancy and sparsity is minimised by using a threedimensional data cube instead of a simple twodimensional table. All the records or clauses and their linguistic analyses can then be combined into this single data structure containing more than two dimensions or a "data cube".

Such a language-oriented, multidimensional database of the linguistic characteristics of the Hebrew text of the Old Testament can enable researchers to do *ad hoc* queries. For example, a researcher may want to do a specific search in order to find good examples of a certain syntactic structure, or to explore the mapping of semantic functions onto syntactic functions. Once the data is stored in a properly structured database, this type of query becomes executable.

XML, a subset of SGML, is a suitable technology for transforming free text into a database. "There is a growing need to annotate a text or a whole corpus according to multiple information levels, especially in the field of linguistics. Language data are provided with SGML-based markup encoding phonological, morphological, syntactic, semantic, and pragmatic structure analyses" (Witt et al., 2005: 103). In such an XML implementation a clause's word order can be kept intact, while other features such as syntactic and semantic functions can be marked as elements or attributes. The elements or attributes from the XML "database" can be accessed and processed by a third generation programming language, such as Visual Basic 6 (VB6). A threedimensional array is probably the most effective programming tool for processing the data (see Chapters 2, 3 and 6). An alternative option could be the use of an XML query language (cf. Bourret, 2003; Deutsch et al., 1999).

This chapter will focus on the following aspects:

- Why should XML be explored as an option to build an exploitable database of linguistic data? (See Section 4.5.)
- How can XML be used to build an exploitable linguistic data cube? (See Section 4.6.)
- How can XML represent the syntactic and semantic analyses of free text? (See Section 4.7.)
- How can XML represent inherently multidimensional data? (See Section 4.8.)

However, before these questions can be answered, it is necessary to provide some background on linguistic databases and computational linguistics in general, as well as on the various linguistic layers that could be analysed and the basic building blocks that form the backbone of such a database (see sections 4.2-4.4).

4.2 Linguistic databases and computational linguistics

Researchers who study natural language processing (NLP) may wonder if a project that studies the use of XML to develop a databank of linguistic data should be regarded as proper computational linguistics since it cannot understand, create or translate human language. However, it should be remembered that, according to Wintner (2004: 113), computational linguistics do not only include "the application of various techniques and results from linguistics to computer science" (NLP), but also "the application of various techniques and results from computer science to linguistics, in order to investigate such fundamental problems as what people know when they know a natural language, what they do when they use this knowledge, and how they acquire this knowledge in the first place". A linguistic database[60]

[60] The adjective *linguistic* in the term *linguistic database* here refers to the linguistic content of the database. Jeong & Yoon (2001) use the same term, but apparently refer to the textual design of the database itself, regardless of the content. However, they do not supply a clear definition for the term. It could also refer to their proposed manipulation language. Other authors, such as Buneman et al. (2002: 480), use the term to refer to the content of the database as it is done in this thesis. Petersen (1999: 10) uses the term "text databases" for databases that store texts together with

captures and manipulates human knowledge of language, thus focusing on the first one of these basic issues in the second category (what people know about a language). This part of computational linguistics could perhaps be called *natural language information systems (NLIS)* because it is similar to the application of information technology to business data, studied in Information Systems discipline, of which databases form an integral part. NLIS can improve the storage, extraction, manipulation and exploration of linguistic data. It is, however, not only an end in itself, since tagged corpora are also needed as tools to train natural language processing systems (Wintner, 2004: 131).

Knowledge representation of human language, of which the tagging of documents is a part, is an interdisciplinary methodology that combines the logic and ontology of linguistics with computation (Unsworth, 2001).[61] Like databases, mark-up is a substitute or surrogate of something else (in this case the covertly structured text)[62], which enables the researcher to make his/her assumptions explicit, to test these hypotheses and to derive conclusions from it (cf. ibid.). The names of the tags, attributes and elements used for the mark-up reflect the researcher's "set of ontological commitments" (cf. ibid.). Since any knowledge representation is a fragmentary theory of intelligent reasoning, it should be accepted that no knowledge representation system can capture all the forms of "intelligent reasoning about a literary text" (cf. ibid.).

This study is limited to the study of word groups, syntactic and semantic functions, excluding other perspectives such as morphology and pragmatics. A simplified version of the semantic functions, according to the functional grammar theory of SC Dik (1997a, 1997b) was used for the semantic analysis. Equally simple systems, compiled by the author, were used for the word-group and syntactic analyses. The reader’s own views may differ from the analyses given here, but it should be kept in

linguistic analyses of it (that is, *expounded text* vs. *text-dominated databases* that are composed mainly by means of characters).

[61] Compare Huitfeldt's (2004) opinion that the semantic web lies "at the intersection of markup technology and knowledge representation".

[62] "Semiotic and linguistic forms are incoherent because they have to be marked in order to be perceived at all" (McGann, 2003: 5).

mind that the main focus of this project is not defining a linguistic theory, but rather illustrating the digital storage and processing of text analyses. Any other linguistic system may be used as the theory underlying the analysis and tagging.

4.3 Linguistic layers

Witt (2002) suggests that various levels of linguistic data could be annotated in separate document grammars, which can be integrated via computer programs. He proposes *i.a.* morphology, syntax and semantics as levels to be annotated: "For the annotation of linguistic data this [i.e. a single level of annotation - JHK] could be e.g. the level of morphology, the level of syllable structures, a level of syntactic categories (e.g. noun, verb), a level of syntactic functions (e.g. subject, object), or a level of semantic roles (e.g. agent, instrument)."

In a later article, Witt (2005: 57) differentiates between linguistic levels and layers. *Levels* refer to divergent logical units such as text layout versus linguistic analyses, and *layers* or *tiers* refer to the various possibilities on one level (for example, syntactic and semantic functions, which are structures that order the text hierarchically). In this study the terms *layers* or *modules* are also used to refer to the various perspectives of syntax, semantics, etc. (cf. Chapters 2 and 3). However, the distinction between *level* and *layer* is not strictly maintained in references to other authors' work, where the terms are used as synonyms. T. Sasaki (2004: 22), for example, uses the term *level* to refer to various linguistic annotations of text, i.e. syntactic, morpho-syntactic, lexical and morphological annotation. It should, however, not cause much misunderstanding, since this study focuses only on one "logical unit", the linguistic analyses, while the verse numbers are only used for primary keys and referencing.

Furthermore, the reader should note that linguists do not necessarily use the names of language modules in exactly the same way. For example, Witt's syntactic categories are the same as Sasaki's morpho-syntactic categories (part-of-speech tagging), while morpho-syntax is used in the current study to refer to word groups.

The use of these terms is theory-bound and the user of a linguistic database should make sure that he/she knows the specific definitions used in a particular implementation. (See Addenda C – F for an overview of the phonetic transcription system, and taxonomies of phrase types, syntactic functions and semantic functions used in this study.)

4.4 The phrase as basic building block of the database structure

The problems of redundancy and sparsity were discussed above and it was indicated that using the phrase as the basic building block of structure for a clause cube may minimise these problems. This solution is discussed in more detail in this section.

Witt (2002) proposes that linguistic database creators use the basic written text as a link, which he calls the *primary data,* between the layers: "… when designing the document grammar it is necessary to consider that the primary data is the link between all layers of annotation". The simplest way to deal with such an implementation is to mark up the various layers of linguistic analysis in separate documents, using the primary data to interrelate the information contained in these documents. Even if the information of all analysed layers are merged into one data structure, such as a data cube, it is still logical to use the basic text (divided into words or phrases), as the basic elements to which all other layers are related.

Depending on the characteristics of the layers to be annotated one should decide whether to use letters, words, phrases, etc., as the reference units. Compare Witt (2005: 65, 70, 72): "… in larger text single words could serve as the reference units" (as opposed to single letters in smaller text). For example, in a project that aims to study morphological analysis it would be necessary to use characters as the smallest units (Bayerl et al., 2003: 165). In this project phrases or word groups are used as the unit of reference.[63] It is, however, important to note that annotations that use

[63] See the parallel discussion in Chapter 2 (2.3, 2.6) where the same concepts are discussed in terms of array technology. In this chapter the focus is on the implementation in an annotated, XML databank.

different units of reference cannot easily be integrated if the text is used as the primary data (the "implicit link" between the layers). This could be solved by numbering the smallest units to be analysed and by referring to the various combinations of these numbers for the divergent layers of analysis (compare Petersen, 1999: 13-14).[64] Although different solutions were researched for the representation of divergent linguistic analyses, "[t]he annotation of multiple hierarchies with SGML-based markup systems is still one of the fundamental problems of text-technological research" (Witt et al., 2005:103). Although this is not a problem in the experiment of this project, it should be researched if one would have to integrate a word group-based analysis with other studies based on letters, morphemes, words or other different units of structure. Compare, for example, Petersen (2004b) who uses words in their original order as the basic units of reference in his textual database. He does, however, add a numbering system to facilitate the mapping of non-congruent linguistic layers.

4.5 Why should XML be explored as an option to build an exploitable database of linguistic data?

The sections above have clearly indicated why it is desirable to build a linguistic database for capturing data regarding the various linguistic layers of text using the phrase as a basic unit of structure. The ideal solution is to keep the database separate and independent from the program(s) that operate on it in order to avoid structural dependence and data dependence. Structural dependence refers to the situation where changing the structure of the databank necessitates all access programs to be adapted, while data dependence refers to a "condition in which data representation and manipulation are dependent on the physical data storage characteristics" (Rob & Coronel, 2007: 15, 640, 652). Therefore, it is not ideal to implement the databank as a module within the VB6b program (as it was done in Chapters 2 and 3).

[64] The basic elements (for example, letters or words) are numbered in order of appearance using integers called *monads* (Petersen, 1999: 13).

This section focuses on the choice of XML to implement a structure-independent and data-indepedent solution. Storing the clause-cube data in a separate, platform-independent, XML file, will make the data available to be used and reused by various access programs. If the structure or content of either the progam or database changes, only the interface between the two needs to be adapted to read the data to and from the threedimensional array, a procedure which will be discussed in the next chapter.

The research question in the heading of this section ("Why should XML be explored as an option to build an exploitable database of linguistic data?") can be broken down into four sub-questions, which will be discussed below:

- Why is XML suitable for implementing a database?
- Why is XML suitable for linguistic data?
- Why is XML suitable for data exploration?
- What are the disadvantages of XML?

4.5.1 Why is XML suitable for implementing a database?

The idea for this study originated while working on an earlier project about the use of HTML to represent linguistic data in a table format (Kroeze, 2002). The tables used in HTML prompted the idea to capture the data in a database, but also showed the limitations of HTML because the tags are only used for formatting and do not contain any semantic information which can be used for structuring purposes.[65] XML, on the other hand, allows the designer of the software to define his/her own tags which may be organised in a hierarchical manner to structure the data.[66] This built-in structure can be used, not only to visualise the data in a way similar to the HTML tables referred to above, but also to process the data for more advanced functionality.

[65] As is the case with unstructured web data, the lack of structure facilitated by HTML causes serious limitations on information access (Xyleme, 2001: 1).

[66] Relational databases use tables or flat structures while XML uses a hierarchical structure that is "arbitrarily deep and almost unrestrictedly interrelated" (Smiljanić et al., 2002: 9).

The hierarchical nature of XML is a major benefit in comparison to simple relational database management systems that make use of collections of flat, twodimensional tables. Use of this technology would lead to sparsity and redundancy problems (see above).[67] Although more complex types of relational database technology exist that do facilitate multidimensional tables, which could provide alternative solutions for multidimensional linguistic data, this study is limited to the investigation of the use of XML as a solution.

The database facilities of XML can be ascribed to its features of allowing the design of unique tag sets and the separation of formatting and structure. A unique set of tags (schema), which fits the relevant data set in a natural way (Flynn, 2002: 56), can be compiled to be the equivalent of a database structure. The structuring is built into a well-designed mark-up schema, but the formatting is covered by separated style sheets. While the schema of a relational database management system exists separately from the data, in XML it coexists with the data as element names or "tags" (Deutsch et al., 1999: 1156). One of the benefits of "the deferral of formatting choices" includes the facilitation of consistent formatting and avoidance of many opportunities for data corruption (DeRose et al., 1990: 15, 17).

Although XML is very suitable for storing data, it should, however, be remembered that the CRUD functions (create, retrieve, update, delete) are actually not done by the XML document itself but by another program that operates on the data in the XML file. Maybe one should even consider the possibility of rather using the term XML data*bank* rather than data*base*: "An XML document is a database only in the strictest sense of the term" because it is essentially only a simple file containing data, organised in a linear fashion (Bourret, 2003). Combined with its surrounding technologies XML may be regarded as a database system, albeit in the "looser sense of the term" because it does provide some of the typical functionalities of "real databases" but also lacks others (ibid.). However, in conventional database terminology, database refers to the collection of tables containing related data,[68] database management system refers to the program that enables creation, reading,

[67] Storing XML data in conventional databases is not ideal since it "artificially creates lots of tuples/objects for even medium-sized documents" (Xyleme, 2001: 3).

[68] Or *static* database – a database without CRUD facilities (cf. Petersen, 1999: 11).

updating and deletion of data in the database, and database system is the combination of a database and the software used to manage it (Smiljanić et al., 2002: 8). In a database approach one may "consider an XML document to be a database and a DTD to be a database *schema*" (Deutsch et al., 1999: 1155). Therefore, in this experiment the XML document refers to the database, the VB6 program may be regarded as a (simple) database management system, and the combination as a database system.

Although it is not implemented in this experiment, using XML to structure the data in the clause cube could facilitate the request and delivery of information through the world wide web in a similar way as is the case with business data. Huang & Su (2002), for example, combine XML technology and push and pull strategies to provide users via the Internet only with information relevant to them. Because an XML document is text-based it is ideal for storage and delivery of business data via the web, which requires a onedimensional stream of characters for efficient transfer. This text-based property of XML also renders it quite suitable for the storage and transfer of linguistic data over the Internet.

4.5.2 Why is XML suitable for linguistic data?

Since XML itself is text based, it follows that it should provide a suitable way to capture textual data. The source text can be kept intact while additional information is added by means of semantic mark-up. Since humanities scholars do not only use texts to transmit information about other phenomena, but also study the texts themselves, it is important to preserve these texts in a form that will facilitate future research. XML provides a way to store both the original text and the results of research on it for future reuse (Huitfeld, 2004). Due to its widespread use and adaptability to other software packages, Flynn (2002: 59) regards XML as the future "*lingua franca* for structured text in the humanities and elsewhere". XML was also recommended by the E-MELD project as a mark-up language in order to create a common standard for and sharing of digital linguistic data (Bird et al., 2002: 432).

XML uses terms to describe texts that are not linked to a specific formatter, such as those suggested by the OHCO model (ordered hierarchy of content objects), and therefore makes documents transportable (platform-independent) (DeRose et al., 1990: 15). "It is a non-propriety public standard independent of any commercial factor and interest" (T. Sasaki, 2004: 19).

According to T. Sasaki (2004:18) researchers of Hebrew linguistics "can benefit enormously" from the use of XML as a medium to store and interchange their research data. An XML database that captures human linguistic analyses and facilitates data warehousing and data mining procedures[69] on this data, for example, could be very helpful to fill the gaps that cannot yet be covered by algorithms that simulate the complex processes of human language. Due to the ambiguity of human language on various layers of phonology, morphology, syntax, semantics and pragmatics, natural language processing systems are not satisfactorily successful, especially on the higher layers of language understanding (Wintner, 2004: 114-118).[70] In fact, such a database can also provide more basic data that can be used to improve NLP systems.

XML is a very scalable medium for storing linguistic data. It is very easy to embed another layer into the hierarchical structure to capture additional information. Besides capturing data that pertains to the text itself, information about parallel texts can be represented in the same manner, thus enabling textual criticism (the process of comparing various editions of a text in order to reconstruct the original text).[71] In this regard, Aarseth (*s.a.*) is very positive about the prospects of hypertext technology: "Not only does hypertext promise a tool for critical annotation and the representation of intertextuality, as well as a useful method for representing complex editions of

[69] "Data Warehousing and Knowledge Discovery technologies are emerging as key technologies to improve data analysis ... and automatic extraction of knowledge from data" (Wang & Dong, 2001: 48).

[70] Even using semantic information in a dictionary does not guarantee the correct interpretation because a machine's interpretation "does not [always] fit conditions in the real world" (Ornan, 2004).

[71] Due to the stability of the text of the Hebrew Bible it is not necessary to consider the use of change-centric management of the XML clause cube, which only contains analyses of a single version of the text. However, in text-critical projects of the text such an approach could be useful for users to obtain snapshots of the text's history (cf. Marian et al., 2001).

variorum texts, it also has become, for many, an incarnation of the post-structural concept of text."

Word order is an important and often essential characteristic of language. In a database that captures linguistic analyses according to logically organised attributes (for example, subject, object, indirect object), the word order is lost and another field is needed for every word to register its word order position. However, XML's simple linear file characteristic makes it very suitable for textual databases since text is also ordered in a linear fashion. It allows the designer to keep the word order intact and to capture the analytical data by means of mark-up. Not only does this eliminate the need for a word-order field, but it also reduces processing to rebuild the original text for output purposes.

Like SGML,[72] XML can be used to annotate either more text-oriented documents or more data-oriented documents.[73] It is therefore very suitable for a linguistic data cube, which is something in between. On the one hand, the text and word order is preserved,[74] and on the other hand, the database is structured to such an extent that it can be represented by a threedimensional array in VB6. This could, therefore, serve as an example where the boundaries between document-centric and data-centric XML documents are blurred (cf. T. Sasaki, 2004: 19).[75]

The characteristics of XML discussed above make it very suitable to record linguistic data, for example in a data cube. In combination with a suitable program this data can be read, updated and deleted in various combinations. A data mart could be built

[72] Cf. DeRose et al. (1990: 12): "It [SGML – JHK] does not prejudice whether a document is to be treated as a database, a word-processing file, or something completely different".

[73] A dictionary is a typical example of a data-oriented linguistic document (cf. Bird et al., 2002).

[74] This statement has to be qualified somewhat. Embedded phrases and clauses challenged the ideal to exactly reproduce the original word order. A compromise was to refer to these embedded elements by using square brackets where they do occur and to analyse them separately afterwards as individual phrases or clauses.

[75] Document-centric documents are also called narrative-centric or text-centric documents. They "are not so well structured and are meant more for human consumption, while data-centric documents ... are more rigidly structured and meant mainly for machine consumption" (T. Sasaki, 2004: 19).

to summarise subsets of the data, thus enabling advanced processing and retrieval. The following section will discuss the data exploration facilities in more detail.

4.5.3 Why is XML suitable for data exploration?

An XML database facilitates complex searches, for example where two or more conditions are to be true (DeRose et al., 1990: 17). Without a proper database these are done partly manually: the researcher finds all texts that satisfy one condition and then searches within that data for the other conditions. A good program or query language could automate the process of searching for data on more than one parameter within an XML document. It could also facilitate text comparison and the display and correlation of various translations of a text, provided that this data are captured in the XML database (DeRose et al., 1990: 18). This will make the task of a translator or exegete a lot easier by integrating the data from various texts and translations into a single tool.

Data integration from various sources is a typical data warehousing activity. Data marts and data warehouses are often used to integrate and aggregate business data. XML schemas can also be used to interoperate legacy databases when migrating and integrating them into newer databases (Thuraisingham, 2002: 190). XML and its surrounding technology can provide similar benefits for humanistic studies since the OHCO model, for example, facilitates the integration of "a wide variety of different types of data or media into a 'compound document'" (DeRose et al., 1990: 17). The suitability of XML to integrate data from various sources has been demonstrated over and over again. Mangisengi et al. (2001: 337) go one step further in their project to virtually co-locate data warehouse islands using XML as a basis to realise the interoperability of these sources.[76] By not having to physically replicate data into a new enormous data warehouse they ensure an efficient load balance. This demonstrates the scalability of projects built on XML technology. (Compare Chapter

[76] According to Wang & Dong (2001: 51) a data warehouse is "a finite set of documents (or data cubes) conforming to one of the XML schema definitions in meta data." A data warehouse is actually a collection of data marts that contain aggregated data.

3 for a more detailed discussion of typical data warehousing procedures facilitated by a clause cube.)

Having a data warehouse is an important step towards efficient data exploration or data mining. Data mining is the process of discovering hidden patterns within large datasets. "The OHCO model treats documents and related files as a database of text elements that can be systematically manipulated …. full-text searches in textbases can specify structural conditions on patterns searched for and text to be retrieved" (DeRose et al., 1990: 17). The location of patterns is the essence of humanistic inquiry which presumes an openness on the side of the researcher, and "databases are perhaps the most well suited to facilitating and exploiting" this enterprise (Ramsay, *s.a.*). It should be noted that data mining is not a coincidental process of discovery, but rather a deliberate process of knowledge invention and construction (cf. Du Plooy, 1998: 54, 59).

4.5.4 What are the disadvantages of XML?

In comparison to all these benefits of XML there are only a few disadvantages (cf. T. Sasaki, 2004: 19). The XML documents can become rather large since the tags are repeated over and over again for each element. In the clause cube experiment of this project, not only the tags but also the character data is used repetitively because the word groups, syntactic functions and semantic functions are encoded as text elements. This design is, however, very suitable for the eventual conversion to an array structure in VB6. According to Buneman et al. (2002: 475) an XML document may be regarded as a hierarchical structure of elements, attributes and text nodes, of which only "[t]ext and element children are held in what is essentially an array".

In a later version of this project the size of the XML document(s) may be reduced dramatically by defining the names of syntactic and semantic functions as entities (for example, <!ENTITY Ben "Beneficiary">) and using repetitive entity references in the database (for example, &Ben;) instead (cf. Burnard, 2004). This provides a viable alternative to compressing techniques to reduce the size of an XML document since "lossy" compression techniques are more suitable for database-like documents, and "lossless" compression techniques are not nearly as efficient as "lossy" techniques (Cannataro et al., 2001: 3).[77]

Besides the verbosity and repetitiveness, "access to the data is slow due to parsing and text conversion" (Bourret, 2003). On the other hand, in the case of text databases, an XML implementation can actually be quite fast since whole documents are stored together and logical joins are not needed (ibid.).

If the XML code is typed using a basic text editor such as Notepad, it can be annoying and error-prone to type repetitive tags and elements, but if the file is created by electronic means, or by using special XML editors, this problem can be avoided.

The separation of data and formatting provides certain benefits as discussed above, but necessitates the creation of a separate style sheet to inform a web browser, such as Opera or Firefox,[78] how to display the text in the XML document (Flynn, 2002: 57). This is, however, a small price to pay for the database-like benefits provided by the same characteristic and the option to design different formats to suit unique requirements.

In addition, Huitfeld (2004) mentions the following weaknesses of XML: poor support for documents enriched by multimedia, absence of well-defined semantics, and the inherent inadequacy to express overlapping hierarchies which have to be bypassed by artificial means. Since XML itself does not contain semantics, it is important to add semantic content to mark-up in order to enable the study of the ontology it reflects (cf. F. Sasaki, 2004: 3).[79]

[77] During "lossy" compression the document structure is changed and the original document cannot be reproduced by reversing the process. If the compression is lossless the compressed data can be decoded to provide a document that is identical to the original (Cannataro et al., 2001: 2).

[78] Internet explorer does not render the tables, defined in this project's XML style sheet, correctly.

[79] *Mark-up semantics* studies "the formal description of the meaning of document grammars and instance documents", while *semantic markup* "is the addition of semantic information to markup" (F. Sasaki, 2004: 3).

In comparison to the advantages, the disadvantages of XML are rather restricted. Thus, one may conclude that it provides suitable technology to build a linguistic database which can be explored to construct new knowledge.

4.6 How can XML be used to build an exploitable linguistic data cube?

XML is not restricted to a predefined set of static mark-up formulas. The user may define his/her own tags to mark up the relevant text in a suitable way. Therefore, tags, elements and attributes can be designed according to the linguistic paradigm within which the researcher works. XML is also very flexible: it is possible and acceptable to map all properties to elements and child elements (Bourret, 2003), and in this experiment it was actually better to code all the linguistic information as primary data (most basic textual elements) to properly implement the threedimensional data cube concept.[80] Primary data is "simple element types" (Bourret, 2003), which is usually used exclusively for the basic text itself,[81] but XML allows the user to creatively design the structure of the database using the various building blocks available. This is called a tag-based approach versus an attribution-based one. While the attribution-based approach is more readable, the tag-based approach is more expandable and suitable for the representation of multidimensional and hierarchical data (Jeong & Yoon, 2001: 834). Using a tag-based approach to build a linguistic data cube in combination with a VB6 access program will provide a custom-made, but flexible and expandable database management system that is both efficient and user-friendly. It is, of course, very important to use these constructs in a consistent manner. The need to reuse data intelligently (for example, for text mining) depends on a "well-planned tagging scheme" (DeRose et al., 1990: 18). To

[80] Compare T. Sasaki's (2004: 42) example of an entry in a data-centric lexical database of Modern Hebrew where all the mark-up is also done as elements and child elements, without using attribute values. According to Deutsch et al. (1999: 1156) "[s]tructured values are called *elements*".

[81] Compare, for example, T. Sasaki (2004: 29-30). See Huitfeldt (2004): "An SGML document therefore has a natural representation as a tree whose nodes represent elements and whose leaves represent the characters of the document."

facilitate this process, schema languages are available to define the structure of the database and to test the contents of the database to ensure that all entries satisfy the schema rules (cf. T. Sasaki, 2004: 18).

4.7 How can XML represent the syntactic and semantic analyses of free text?

The designer has to think about the data structure as a threedimensional object having one row for each clause; five (in the case of Genesis 1:1-2:3) columns per clause, one for each phrase; and various layers of analysis, *i.a.* one to capture syntactic information and another to record semantic functions. If a phrase does not have a semantic function, for example in the case of conjunctions, an empty value (-) is inserted into the relevant field. Null values would also indicate the absence of a function, but could cause problems during sorting and importing and exporting the XML file to and from a program (round-tripping[82]). In XML the data cube is represented by a hierarchical structure (see below). It is important to validate the recorded data to ensure the consistent use of terminology. A proper XML schema enforces consistency and the proper organization of stored text which is necessary because "[n]o hardware improvements or programming ingenuity can completely overcome a flawed representation" (DeRose et al., 1990: 4). The creation and use of an XML schema will be discussed in more detail below (4.11). In addition, validation of syntactic and semantic functions will also be done by the VB6 program to ensure clean data before advanced processing will be done (see Chapter 6).

A schema is actually a knowledge representation or an ontology[83] that is formulated, consciously or unconsciously, based on a specific theory of language.[84] "If you want

[82] Round-tripping will be discussed in detail in Chapter 5.

[83] "An ontology is a formal conceptualization of a domain that is usable by a computer. Ontologies ... allow applications to agree on the terms that they use when communicating" (Euzenat, 2001: 21).

[84] The XML schema may be regarded as the blueprint for a linguistic ontology since it provides the framework for "a catalog of the types of things that are assumed to exist in a domain of interest" (Sowa, 2003). Because the types are defined only in human language, it should be regarded as an "informal ontology".

a computer to be able to process the materials you work on, whether for search and retrieval, analysis, or transformation—then those materials have to be constructed according to some explicit rules, and with an explicit model of their ontology in view" (Unsworth, 2001). Various ontologies in linguistic projects reflect the various underlying theoretical paradigms, and one can only hope that these will converge to more standardised systems in future. Divergent ontologies are not optimised to play the role of a "key factor for enabling interoperability in the semantic web" (ibid.) However, one will have to accept that linguistic ontologies are phenomena that evolve in parallel to the underlying philosophies that they reflect; since it is a humanistic field of study, it will never be as rigorous as the natural sciences. XML could at least help the comparison of the various approaches. With reference to literary analysis, McGann (2003: 5) says: "Textuality is, like light, fundamentally incoherent. To bring coherence to either text or to light requires great effort and ingenuity, and in neither case can the goal of perfect coherence be attained." Although "any philosophy is destined to be incomplete", ontologies are important because "[w]ithout it, there is no hope of merging and integrating the ever expanding and multiplying databases and knowledge bases around the world" (Sowa, 2003).

4.8 How can XML represent inherently multidimensional data?

According to Witt (2002) using separate annotated document grammars for the various linguistic layers allows "an unlimited number of concurrent annotations". It would indeed be easier to annotate each layer in a separate XML document, but the use would be very limited. In order to study the mappings of the linguistic layers, for example, one needs an integrated structure because "separate annotations do not allow for establishing relations between the annotation tiers" (Witt, 2002).[85] Even Witt et al. (2005: 105) acknowledge the need to integrate multiple notations into a single XML representation. One could, of course, use a system of primary and foreign keys to join the various annotation tiers of separate documents, but it will cause a lot of overhead. Using a threedimensional data structure instead can eliminate a lot of conversion and programming to merge various XML databases into one. There is a

[85] Also see Witt et al. (2005: 112).

natural similarity between data cubes and XML databases since both are multidimensional and hierarchical in character (Wang & Dong, 2001: 50).

A data cube merges all data in one structure, eliminating a lot of overhead in terms of programming needed for the comparison of separate files and the inference of relations between their elements (cf. Witt, 2005: 56), because the various layers are already interrelated by the threedimensional data structure. It is also unlimited since more layers can be added on the depth axis to capture additional layers of analysis. In this experiment one annotation level (the third dimension) serves several linguistic modules (cf. Bayerl et al., 2003: 164): phonology, translation, word groups, syntax and semantics.

An XML database is of course a text-based document which is essentially onedimensional because text represents a stream of language utterances. Therefore, one should "collapse" the (conceptual) threedimensional data cube into a onedimensional stream of tags and primary data. The tagging structure should represent a consistent hierarchy which can be interpreted by a program to convert the stream of text into a data cube. The structure used in this experiment will be discussed in the next section.

4.9 The structure of the Genesis 1:1-2:3 database in XML

As discussed above, it is very important to design a proper structure for an XML database. "Like relational databases, there is nothing in native XML databases that forces you to normalize your data. That is, you can design bad data storage with a native XML database just as easily as you can with a relational database. Thus, it is important to consider the structure of your documents before you store them in a native XML database" (Bourret, 2003). The hierarchy of the Genesis 1:1-2:3 clause cube is shown in Figure 4.1.

```
Hebrew Bible                    - not used in this study
  Bible Book                    - not used in this study
  Pericope[86]             - root element in this study: <Genesis1v1-2v3>
    Clause                 - each clause represented by one table: <clause>
      Clause Number        - each clause's ID: <clauseno>
      Table Headers        - headings for each column: <headers><header>
      Language Levels 1-5 - the various modules of analysis: <level1> ...
        Level Description - description of module per row: <leveldesc>
        Phrases 1-5        - the word groups in a clause: <phrase1> ...
```

Figure 4.1. The hierarchy of the Genesis 1:1-2:3 clause cube as reflected by its XML implementation.

This hierarchy actually represents various levels and layers. Although other documents could be used to mark up other versions of analyses and the various documents connected by means of the identical textual content, these analyses may also often be combined in a single document - compare Witt et al. (2005: 104, 105): "Sometimes, the single hierarchy restriction is not perceived as a drawback because annotations with concepts from different information levels can often be integrated in a single hierarchy." In the Genesis 1:1-2:3 clause cube the structure of the text (book, pericope, clause, phrase) is mixed in a single hierarchy with the concepts of the linguistic modules (phonology, morpho-syntax, syntax, semantics) since the VB6 management program will use the tag structure to convert the rather flat XML file to build the threedimensional clause cube as a threedimensional array.

The XML schema which describes the structure of the XML database is based on the logical hierarchical structure. An example of an XML schema to annotate text, focusing only on the structure of the text, can be found in Witt et al. (2005: 105). It contains the hierarchy shown in Figure 4.2.[87]

[86] In this experiment Genesis 1:1-2:3, the first pericope of the Hebrew Bible, is used as the basic text and root element. Although it could be argued that Genesis 2:4a also belongs to this pericope, it was decided not to include this clause, following the masoretic division. If a longer text were used as corpus, one would have to decide whether the segmentations on this level should be done by chapter or pericope.

[87] Compare T. Sasaki (2004: 23) for a similar, but different schema of mark-up for a Modern Hebrew corpus. See also Petersen (2004b) and Buneman et al. (2002: 481).

```
<article>
   <section>
      <title> ... </title>
      <paragraph> ... </paragraph>
      <itemizedlist>
         <listitem> ... </listitem>
      </itemizedlist>
   </section>
</article>
```

Figure 4.2. An example of an XML schema used to annotate text (Witt et al., 2005: 105).

This concept can be expanded to cover more than one level of analysis by using the hierarchy of structural and analytical elements above in the design of the structure of the XML database of Genesis 1:1-2:3, as shown in Figure 4.3 below. The five phrases per clause that have been used as the structuring backbone are sufficient for Genesis 1:1-2:3, but may have to be extended for other texts (see 4.4 above). The five linguistic layers that have been chosen here, are sufficient to illustrate the multidimensionality of the data structure and may be extended to cover other needs.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
 <Genesis1v1-2v3>
   <clause>
      <clauseno></clauseno>
      <headers>[88]
         <header>Level</header>
         <header>Phrase1</header>
         <header>Phrase2</header>
         <header>Phrase3</header>
         <header>Phrase4</header>
         <header>Phrase5</header>
      </headers>
      <level1>
         <leveldesc>Phon:</leveldesc>
```

[88] One could argue that the repetitive tagging of structural information, such as "Level", "Phrase1", "Phon:", etc., is superfluous. However, it does help to keep the XML file human-readable.

```
         <phrase1></phrase1>
         <phrase2></phrase2>
         <phrase3></phrase3>
         <phrase4></phrase4>
         <phrase5></phrase5>
         </level1>
      <level2>
         <leveldesc>Translation:</leveldesc>
         <phrase1></phrase1>
         <phrase2></phrase2>
         <phrase3></phrase3>
         <phrase4></phrase4>
         <phrase5></phrase5>
      </level2>
      <level3>
         <leveldesc>Phrase type:</leveldesc>
         <phrase1></phrase1>
         <phrase2></phrase2>
         <phrase3></phrase3>
         <phrase4></phrase4>
         <phrase5></phrase5>
      </level3>
      <level4>
         <leveldesc>SynF:</leveldesc>
         <phrase1></phrase1>
         <phrase2></phrase2>
         <phrase3></phrase3>
         <phrase4></phrase4>
         <phrase5></phrase5>
      </level4>
      <level5>
         <leveldesc>SemF:</leveldesc>
         <phrase1></phrase1>
         <phrase2></phrase2>
         <phrase3></phrase3>
         <phrase4></phrase4>
         <phrase5></phrase5>
      </level5>
   </clause>
   <clause> ... </clause>
   <clause> ... </clause>
   <clause> ... </clause> etc.
</Genesis1v1-2v3>
```

Figure 4.3. The basic structure of the XML database of Genesis 1:1-2:3.

When this scheme is populated with linguistic data from Genesis 1:1-2:3, it looks as shown in Figure 4.4 (only the first two clauses are shown below as an example).

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Genesis1v1-2v3>
    <clause>
        <clauseno>Gen01v01a</clauseno>
        <headers>
            <header>Level</header>
            <header>Phrase1</header>
            <header>Phrase2</header>
            <header>Phrase3</header>
            <header>Phrase4</header>
            <header>Phrase5</header>
        </headers>
        <level1>
            <leveldesc>Phon:</leveldesc>
            <phrase1>bre$it</phrase1>
            <phrase2>bara</phrase2>
            <phrase3>elohim</phrase3>
            <phrase4>et ha$amayim ve'et ha'arets</phrase4>
            <phrase5>-</phrase5>
        </level1>
        <level2>
            <leveldesc>Translation:</leveldesc>
            <phrase1>in the beginning</phrase1>
            <phrase2>he created</phrase2>
            <phrase3>God</phrase3>
            <phrase4>the heaven and the earth</phrase4>
            <phrase5>-</phrase5>
        </level2>
        <level3>
            <leveldesc>Phrase type:</leveldesc>
            <phrase1>PP</phrase1>
            <phrase2>VP</phrase2>
            <phrase3>NP</phrase3>
            <phrase4>NP</phrase4>
            <phrase5>-</phrase5>
        </level3>
        <level4>
            <leveldesc>SynF:</leveldesc>
            <phrase1>Adjunct</phrase1>
            <phrase2>Main verb</phrase2>
            <phrase3>Subject</phrase3>
            <phrase4>Object</phrase4>
            <phrase5>-</phrase5>
        </level4>
        <level5>
            <leveldesc>SemF:</leveldesc>
            <phrase1>Time</phrase1>
            <phrase2>Action</phrase2>
            <phrase3>Agent</phrase3>
            <phrase4>Product</phrase4>
            <phrase5>-</phrase5>
        </level5>
    </clause>
    <clause>
        <clauseno>Gen01v02a</clauseno>
        <headers>
            <header>Level</header>
            <header>Phrase1</header>
            <header>Phrase2</header>
            <header>Phrase3</header>
            <header>Phrase4</header>
            <header>Phrase5</header>
        </headers>
        <level1>
            <leveldesc>Phon:</leveldesc>
            <phrase1>veha'arets</phrase1>
            <phrase2>hayta</phrase2>
            <phrase3>tohu vavohu</phrase3>
            <phrase4>-</phrase4>
            <phrase5>-</phrase5>
        </level1>
        <level2>
            <leveldesc>Translation:</leveldesc>
            <phrase1>and the earth</phrase1>
            <phrase2>was</phrase2>
            <phrase3>an emptiness and void</phrase3>
            <phrase4>-</phrase4>
            <phrase5>-</phrase5>
        </level2>
        <level3>
            <leveldesc>Phrase type:</leveldesc>
            <phrase1>NP</phrase1>
            <phrase2>VP</phrase2>
            <phrase3>NP</phrase3>
            <phrase4>-</phrase4>
            <phrase5>-</phrase5>
        </level3>
        <level4>
            <leveldesc>SynF:</leveldesc>
            <phrase1>Subject</phrase1>
            <phrase2>Copulative verb</phrase2>
            <phrase3>Copula-predicate</phrase3>
            <phrase4>-</phrase4>
            <phrase5>-</phrase5>
        </level4>
        <level5>
            <leveldesc>SemF:</leveldesc>
            <phrase1>Zero</phrase1>
            <phrase2>State</phrase2>
            <phrase3>Classification</phrase3>
            <phrase4>-</phrase4>
            <phrase5>-</phrase5>
        </level5>
    </clause> etc.
</Genesis1v1-2v3>
```

Figure 4.4. Two populated clause elements in the XML database.

4.10 Critical discussion of the XML clause cube implementation

The threedimensional cube structure implemented in XML above provides an easy way to resolve identity conflicts, i.e. where elements on the various layers span the same range of words of the basic text (Witt et al., 2005: 107),[89] for example the exact same phrase *et ha$amayim ve'et ha'arec* in Genesis 1:1, which is analysed on the various levels as NP, object and product. The Genesis 1:1-2:3 experiment has many identity conflicts since the basic unit of reference is the phrase (word group). Actually, the whole clause cube structure is built on identity conflicts – in each clause exactly the same phrases are analysed on the various levels. By ignoring conjunctions which are parts of other words (a commonly found phenomenon in Hebrew) it was possible to use exactly the same demarcations for the linguistic modules that were annotated.

[89] "An identity conflict exists when two element instances from the two annotation layers span an identical portion of the text" (Witt et al., 2005: 112).

This structure facilitates the study of mapping between the chosen linguistic modules. The implication of this implementation is that more detailed information, such as morphological analyses (for example, *bre$it* = preposition *be-* + noun *re$it*) cannot be stored by only adding another level on the depth dimension. In order to facilitate functions like these the structure of the clause cube will have to be changed into a more complex structure where words and/or morphemes are numbered, using ranges of the numbers to demarcate phrases on the higher levels of analysis. (Cf. Witt, 2005: 70, for an example of a textual stream where each character has its own, unique identification.) This, however, falls outside the scope of this study.

In a twodimensional representation identity conflicts have to be resolved either by marking up the same texts in various XML files, or by nesting one layer's elements in another layer's elements (cf. Witt et al., 2005: 107).[90] In this project's threedimensional structure, however, the layers are described in parallel structures. In XML these parallel structures are implemented using various collections of elements which are hierarchically on the same layer but separated by descriptive tags. The various collections of sibling and child elements are grouped into units and subunits by wrapper tags.[91] This is a direct representation of the inherently threedimensional data underlying the implementation and avoids the necessity to define some layers as attributes of elements on another layer.

Although one may argue that this is a counter-intuitive implementation of inherently hierarchical linguistic data, it is typical of data-oriented XML files (cf. T. Sasaki, 2004: 31-42).[92] If one implemented the linguistic modules as attributes of the phrases, it

[90] Compare Witt et al. (2005: 109-114) for a discussion of other types of relations (mappings) between various annotated layers, such as inclusion and overlap conflicts (that is, where the parts of the text that are analysed are not exactly the same). Since these types do not occur in this case study they are not discussed further.

[91] Compare T. Sasaki (2004: 32) who also uses a wrapper tag <entry> to organise the various child elements of each lexeme into a unit of a data-centric XML lexical database. A wrapper element is a higher level element used to store multiple "entities" in one XML "table" or various "tables" in one XML database (cf. Bourret, 2003).

[92] Compare T. Sasaki's (2004) example of a data-oriented lexicographical implementation with his example of a document-centric annotation in which the syntactic role is defined as an attribute of a phrase.

would become much more difficult (or even impossible) to build a threedimensional XML cube, since attributes cannot be used for document-structuring purposes, while elements can (Holzner, 2004: 67-68).[93] Lack of structure will have detrimental effects on the advanced processing of the linguistic data (for example studying the mapping of linguistic modules). According to Witt (2005: 55-56), the layers of phonology, morphology, syntax and semantics "are (relatively) independent of each other" – this supports the idea to treat them as separate elements and not as attributes of other elements, a concept which is also mirrored by the threedimensional cube consisting of an array of cells of variables organised according to rows, cell and levels (depth dimensions). In the XML schema the legitimate possibilities of the linguistic levels of morpho-syntax, syntax and semantics are defined as enumerations[94] of element values (see the section on validation below).

One may conclude that the hierarchy of an XML document structure does not, and does not have to, reflect the inherent clause structure. Although the phrases do have syntactic and semantic characteristics or attributes, speaking from a linguistic perspective, these may be implemented in XML as elements for the sake of threedimensional structuring and processing. To define these linguistic attributes as XML elements is, therefore, a pragmatic decision, facilitating the database functionalities needed. This "data-centric application of XML" may be quite different from the more conventional "document-centric" applications – data-centric files, which are usually processed by machines, are much more structured (cf. T. Sasaki, 2004: 19).

The original Hebrew text is not marked up using the Hebrew alphabet. Instead a simple phonological rendering is used (see Addendum C). Therefore, one would need another mechanism to link this product to, for example, the Biblia Hebraica Stuttgartensia (BHS), should the need arise. One solution could be to use standoff

[93] Since both attributes and elements hold data, one could use Holzner's (2004: 67) guideline (i.e. using elements to structure the file, and attributes for additional information) to choose which one should be used. Another reason for using elements rather than attributes is that "using too many attributes can make a document hard to read" (Holzner, 2004: 68).

[94] "An enumeration is a set of labels with values", for example the enumeration syntactic function which has the labels of subject, direct object, indirect object, etc. (cf. Petersen, 2004b).

mark-up,[95] a way of separating mark-up from the original text to be annotated. This would require the original text (BHS) to contain basic mark-up identifying each word with a unique primary key, which could be referenced in the standoff annotation (cf. Thompson & McKelvie, 1997). For example the phrases in Genesis 1:1 could be numbered in the BHS as follows: Gen1v1a1: *bre$it,* Gen1v1a2: *bara,* Gen1v1a3: *elohim,* Gen1v1a4: *et-ha$amayim ve'et ha'arets.* These identifiers may then be used to link the original Hebrew text (in the Hebrew alphabet) with the phonological representation used in the clause cube, in this way making explicit the inherent links between the two texts.

Similar to the procedure in T. Sasaki (2004: 24), only the verbal core is marked as VP.[96] Petersen (2004b) follows a similar approach: in the clause "The door was blue" only the copulative verb is marked as VP.[97] Including other phrases such as complements, direct objects and adverbials in the verb phrase would necessitate another layer of analysis and the distinction of inclusive relationships, which fall outside the scope of this study. However, in this study, preposition phrases are regarded as the combination of the preposition and its complement – this is different from T. Sasaki who regards the preposition phrase as a linking unit between the verb and its satellite (which actually is more consistent and in line with the VP scenario).

In this experiment the names of word groups, syntactic functions and semantic functions could be regarded as foreign keys – these could be used as primary keys in other "tables" or documents where definitions are supplied. This is, however, not implemented in this study. If these documents were created, one would have to ensure referential integrity between the foreign keys and primary keys. Textual child elements referring to word groups, syntactic functions and semantic functions are primary data that must be regarded as external pointers (or foreign keys) which point

[95] Standoff annotation is necessary when the original text is read-only, copyright protected or prompts overlapping hierarchies (Thompson & McKelvie, 1997).

[96] In Functional Grammar a clause (or "predication") is regarded as a combination of a verb with its arguments and satellites (see Dik, 1997a: 77). This is similar to T. Sasaki's principle: "This scheme proposes to annotate syntactic argument structure with verbs as the core and other phrases as their satellites".

[97] Also cf. Ornan (2004).

to valid document fragments in the related documents (cf. Bourret, 2003). One should therefore ensure that the names of these features are used absolutely consistently: it would, for example, be unacceptable to use both *subj* and *Subject* to tag the subject of a clause. Although these foreign key elements will be used over and over again, redundancy is acceptable in the case of foreign keys.

The verse number elements in XML (e.g., <clauseno>Gen01v01a</clauseno>) may be regarded as primary (or candidate) keys that uniquely identify every clause. These keys facilitate searches and references to specific clauses. "If XML documents are to do double duty as databases, then we shall need keys for them" (Buneman et al., 2002: 473). When the clause number is used as a reference to an embedded clause, it functions as a foreign key. It may be coded as part of another phrase and one should be able to find it using a "fuzzy" search (where a query searches for a part of a string appearing within a bigger attribute value). In this case, the verse numbers are considered as internal pointers since they refer to another section of the same document. Relative clauses, for example, are regarded as embedded clauses (EC). The whole clause is referred to in the main clause, and the relative clause is then analysed separately. Other ECs and embedded clause clusters (ECC), such as direct speech, are treated in the same way. The ECs and ECCs are similar to the "gaps" used by Petersen (2004b) in his Emdros project. It may therefore be concluded that the clause cube would have been normalised.[98]

4.11 Validating the XML document

A schema[99] was created using the built-in functionality of Visual Studio.Net 2003 (VS.Net 2003).[100] Although the basic schema was automatically created, three

[98] Normalisation is the process of minimising redundant data in a database (Connolly & Begg, 2005: 390).

[99] The structure of an XML document is represented by its schema. An "XML schema with a lower case 's' refers to any XML schema – such as a DTD, an XML Schema document, or a RELAX NG schema" (Bourret, 2003).

[100] VS2003.Net was used because the XML functionality is not available in VB6. VS2005.Net allows one to automatically create an XML Schema, but not to use it directly to validate XML databases.

simple types and enumerations of phrases tags, as well as syntactic and semantic function tags, were coded manually and added to the schema. A simple type is a user-defined type, which enables the programmer to create custom-made types that reflect his/her exact requirements (Deitel & Deitel, 2006: 919-921); for example, one may create types to define lists (enumerations) of possible values of phrases (word groups) and syntactic and semantic functions. An enumeration is "a set of values that a data item can select from" (Holzner, 2004: 213). The schema (Gen1_InputV15.xsd[101]) is shown in Figure 4.5 below (see also Addendum G on the included CD). The XML database itself (Gen1_InputV15.xml) was created by converting a databank module in VB6 (see Chapter 2 and 3) programmatically into a text file, a procedure to be discussed in the following chapter (see also Addendum H on the included CD). The schema was then used to test the XML database of Genesis 1:1-2:3, and this procedure revealed some inconsistencies in the tagging, for example with regard to the use of square brackets to indicate embedded clauses. After correcting these tagging errors (see Addendum I for the corrected file, Gen1_InputV15b.xml) the validation was successful.

```
<?xml version="1.0"?>
<xs:schema id="Genesis1v1-2v3"
targetNamespace="http://tempuri.org[102]/Gen1_InputV15.xsd"
xmlns:mstns="http://tempuri.org/Gen1_InputV15.xsd"
xmlns="http://tempuri.org/Gen1_InputV15.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-
microsoft-com:xml-msdata" attributeFormDefault="qualified"
elementFormDefault="qualified">

 <xs:simpleType name="WGenum">
  <xs:restriction base="xs:string">
   <xs:enumeration value="AdvP"/>
   <xs:enumeration value="AP"/>
   <xs:enumeration value="ConjP"/>
   <xs:enumeration value="EC"/>
   <xs:enumeration value="ECC"/>
   <xs:enumeration value="NP"/>
```

Enumeration of phrase types as possible elements of a simple type ("WGenum")

VS2003.Net, however, facilitates both automatic creation and direct validation (using an option on the XML menu).

[101] The XSD and XML files can be opened and viewed with Notepad.

```
    <xs:enumeration value="ParticleP"/>
    <xs:enumeration value="PP"/>
    <xs:enumeration value="RP"/>
    <xs:enumeration value="VP"/>
    <xs:enumeration value="-"/>
  </xs:restriction>
  </xs:simpleType>

 <xs:simpleType name="synfenum">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Subject"/>
    <xs:enumeration value="Predicate"/>
    <xs:enumeration value="Main verb"/>
    <xs:enumeration value="Transitive verb"/>
    <xs:enumeration value="Intransitive verb"/>
    <xs:enumeration value="Preposition verb"/>
    <xs:enumeration value="Copulative verb"/>
    <xs:enumeration value="Copula"/>
    <xs:enumeration value="Copula-predicate"/>
    <xs:enumeration value="Complement"/>
    <xs:enumeration value="Object"/>
    <xs:enumeration value="Object clause"/>
    <xs:enumeration value="Object cluster"/>
    <xs:enumeration value="IndObj"/>
    <xs:enumeration value="Copula-predicate"/>
    <xs:enumeration value="Adjunct"/>
    <xs:enumeration value="Attribute"/>
    <xs:enumeration value="Disjunct"/>
    <xs:enumeration value="Interjection"/>
    <xs:enumeration value="Modal word"/>
    <xs:enumeration value="Discourse marker"/>
    <xs:enumeration value="Dislocative"/>
    <xs:enumeration value="Addressee"/>
    <xs:enumeration value="Conj"/>
    <xs:enumeration value="Co-ordinate conjunction"/>
    <xs:enumeration value="Subordinate conjuction"/>
    <xs:enumeration value="Relative particle"/>
    <xs:enumeration value="-"/>
  </xs:restriction>
```

Enumeration of syntactic functions as possible elements of a simple type ("synfenum")

[102] "tempuri.org is the default namespace URI used by Microsoft development products, like Visual Studio. 'tempuri' is short for Temporary Uniform Resource Identifier" (http://en.wikipedia.org/wiki/Tempuri). Namespaces are essential to avoid conflicting sets of tags (Holzner, 2004: 92).

```
  </xs:simpleType>

 <xs:simpleType name="semfenum">
  <xs:restriction base="xs:string">
   <xs:enumeration value="Action"/>
   <xs:enumeration value="Position"/>
   <xs:enumeration value="Process"/>
   <xs:enumeration value="State"/>
   <xs:enumeration value="Agent"/>
   <xs:enumeration value="Positioner"/>
   <xs:enumeration value="Force"/>
   <xs:enumeration value="Processed"/>
   <xs:enumeration value="Zero"/>
   <xs:enumeration value="Patient"/>
   <xs:enumeration value="Product"/>
   <xs:enumeration value="Receiver"/>
   <xs:enumeration value="Location"/>
   <xs:enumeration value="Direction"/>
   <xs:enumeration value="Source"/>
   <xs:enumeration value="Reference"/>
   <xs:enumeration value="Beneficiary"/>
   <xs:enumeration value="Company"/>
   <xs:enumeration value="Instrument"/>
   <xs:enumeration value="Manner"/>
   <xs:enumeration value="Speed"/>
   <xs:enumeration value="Role"/>
   <xs:enumeration value="Path"/>
   <xs:enumeration value="Time"/>
   <xs:enumeration value="Circumstance"/>
   <xs:enumeration value="Result"/>
   <xs:enumeration value="Purpose"/>
   <xs:enumeration value="Reason"/>
   <xs:enumeration value="Cause"/>
   <xs:enumeration value="Existence"/>
   <xs:enumeration value="Identification"/>
   <xs:enumeration value="Classification"/>
   <xs:enumeration value="Quality"/>
   <xs:enumeration value="Posessor"/>
   <xs:enumeration value="-"/>
  </xs:restriction>
  </xs:simpleType>

 <xs:element name="Genesis1v1-2v3" msdata:IsDataSet="true"
msdata:Locale="en-ZA" msdata:EnforceConstraints="False">
 <xs:complexType>
  <xs:choice maxOccurs="unbounded">
```

Enumeration of semantic functions as possible elements of a simple type ("semfenum")

```
  <xs:element name="clause">
   <xs:complexType>
   <xs:sequence>

    <xs:element name="clauseno" type="xs:string" minOccurs="0" />
    <xs:element name="headers" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
     <xs:sequence>
     <xs:element name="header" nillable="true" minOccurs="0"
maxOccurs="unbounded">
      <xs:complexType>
      <xs:simpleContent msdata:ColumnName="header_Text" msdata:Ordinal="0">
       <xs:extension base="xs:string">
       </xs:extension>
      </xs:simpleContent>
      </xs:complexType>
     </xs:element>
     </xs:sequence>
    </xs:complexType>
    </xs:element>

    <xs:element name="level1" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
     <xs:sequence>
     <xs:element name="leveldesc" type="xs:string" minOccurs="0" />
     <xs:element name="phrase1" type="xs:string" minOccurs="0" />
     <xs:element name="phrase2" type="xs:string" minOccurs="0" />
     <xs:element name="phrase3" type="xs:string" minOccurs="0" />
     <xs:element name="phrase4" type="xs:string" minOccurs="0" />
     <xs:element name="phrase5" type="xs:string" minOccurs="0" />
     </xs:sequence>
    </xs:complexType>
    </xs:element>

    <xs:element name="level2" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
     <xs:sequence>
     <xs:element name="leveldesc" type="xs:string" minOccurs="0" />
     <xs:element name="phrase1" type="xs:string" minOccurs="0" />
     <xs:element name="phrase2" type="xs:string" minOccurs="0" />
     <xs:element name="phrase3" type="xs:string" minOccurs="0" />
     <xs:element name="phrase4" type="xs:string" minOccurs="0" />
     <xs:element name="phrase5" type="xs:string" minOccurs="0" />
     </xs:sequence>
    </xs:complexType>
    </xs:element>
```

```
<xs:element name="level3" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
     <xs:sequence>
     <xs:element name="leveldesc" type="xs:string" minOccurs="0" />
       <xs:element name="phrase1" type="mstns:WGenum" minOccurs="0" />
     <xs:element name="phrase2" type="mstns:WGenum" minOccurs="0" />
     <xs:element name="phrase3" type="mstns:WGenum" minOccurs="0" />
     <xs:element name="phrase4" type="mstns:WGenum" minOccurs="0" />
     <xs:element name="phrase5" type="mstns:WGenum" minOccurs="0" />
     </xs:sequence>
    </xs:complexType>
    </xs:element>
```

Using simple type *Genum* to validate phrase elements

```
    <xs:element name="level4" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
     <xs:sequence>
     <xs:element name="leveldesc" type="xs:string" minOccurs="0" />
     <xs:element name="phrase1" type="mstns:synfenum" minOccurs="0" />
     <xs:element name="phrase2" type="mstns:synfenum" minOccurs="0" />
     <xs:element name="phrase3" type="mstns:synfenum" minOccurs="0" />
     <xs:element name="phrase4" type="mstns:synfenum" minOccurs="0" />
     <xs:element name="phrase5" type="mstns:synfenum" minOccurs="0" />
     </xs:sequence>
    </xs:complexType>
    </xs:element>
```

Using simple type *synfenum* to validate syntactic function elements

```
    <xs:element name="level5" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
     <xs:sequence>
     <xs:element name="leveldesc" type="xs:string" minOccurs="0" />
     <xs:element name="phrase1" type="mstns:semfenum" minOccurs="0" />
     <xs:element name="phrase2" type="mstns:semfenum" minOccurs="0" />
     <xs:element name="phrase3" type="mstns:semfenum" minOccurs="0" />
     <xs:element name="phrase4" type="mstns:semfenum" minOccurs="0" />
     <xs:element name="phrase5" type="mstns:semfenum" minOccurs="0" />

     </xs:sequence>
    </xs:complexType>
    </xs:element>
   </xs:sequence>
   </xs:complexType>
  </xs:element>
  </xs:choice>
 </xs:complexType>
 </xs:element>
```

Using simple type *semfenum* to validate semantic function elements

```
</xs:schema>
```

Figure 4.5. The XML Schema used to validate the XML database of Genesis 1:1-2:3.

4.12 Viewing the XML file in a web browser

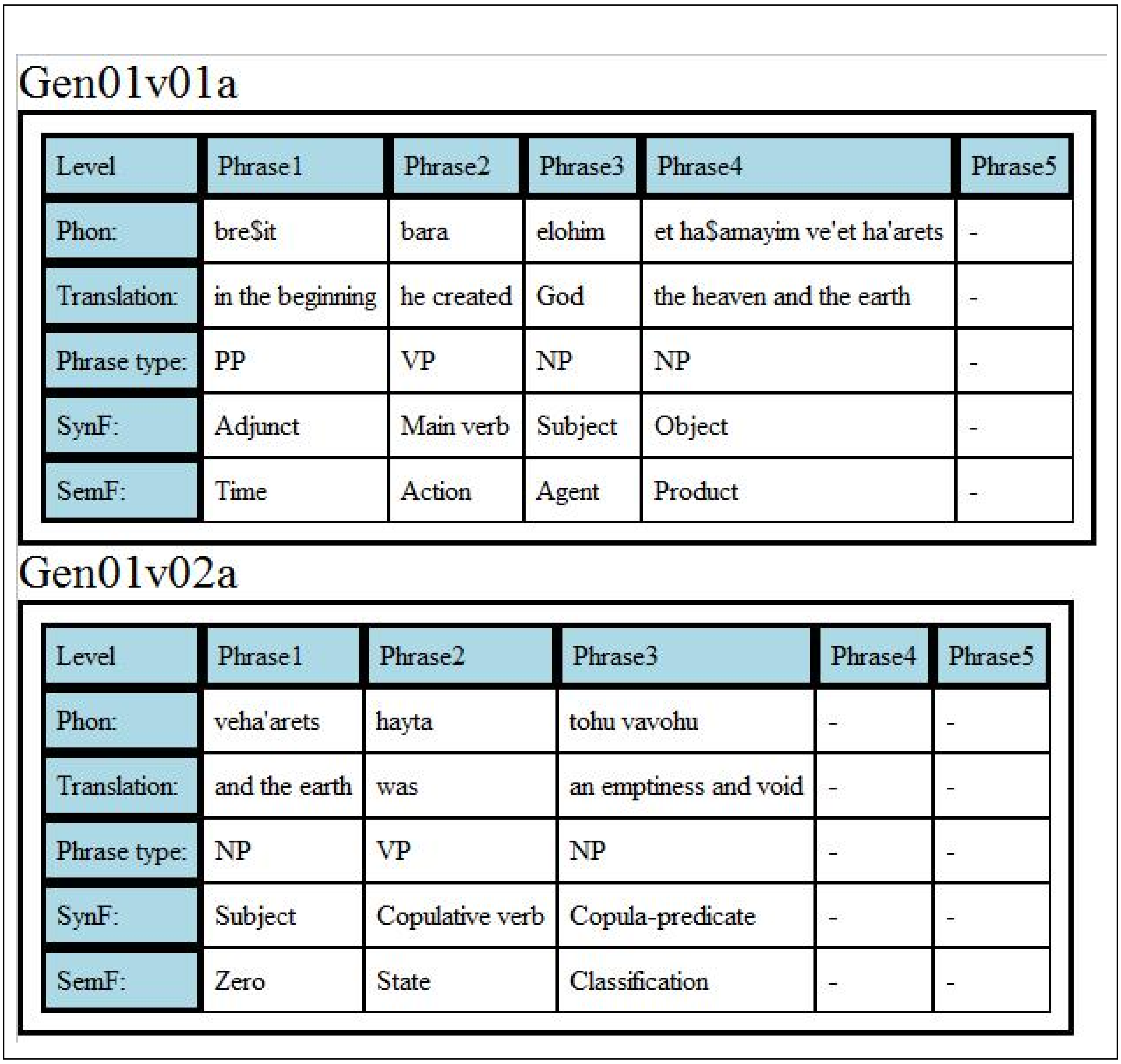
The data in the XML data cube can be visualised in a web browser as a series of twodimensional tables using the style sheet shown in Figure 4.6 (see Addendum J: Gen1XMLdb03c.css).

```
clause {display:table; border-style:solid; margin-top:20; margin-left:20;
     padding:10px}

clauseno {display:table-caption; font-size: 20pt}

headers {display:table-header-group}


header {display:table-cell; padding:6px;
    background-color:lightblue; border-style:solid}

level1 {display:table-row}

level2 {display:table-row}

level3 {display:table-row}

level4 {display:table-row}

level5 {display:table-row}

leveldesc{display:table-cell; background-color:lightblue; border-
     style:solid; border-top-width:medium; border-bottom-width:medium;
     border-left-width:medium; border-right-width:medium; padding:6px;}

phrase1 {display:table-cell; border-style:solid; border-top-width:thin;
     border-bottom-width:thin; border-left-width:thin; border-right-
     width:thin; padding:6px;}
```

```
phrase2 {display:table-cell; border-style:solid; border-top-width:thin;
     border-bottom-width:thin; border-left-width:thin; border-right-
     width:thin; padding:6px;}

phrase3 {display:table-cell; border-style:solid; border-top-width:thin;
     border-bottom-width:thin; border-left-width:thin; border-right-
     width:thin; padding:6px;}

phrase4 {display:table-cell; border-style:solid; border-top-width:thin;
     border-bottom-width:thin; border-left-width:thin; border-right-
     width:thin; padding:6px;}

phrase5 {display:table-cell; border-style:solid; border-top-width:thin;
     border-bottom-width:thin; border-left-width:thin; border-right-
     width:thin; padding:6px;}
```

Figure 4.6. The XML style sheet used to display the XML clause cube as a series of twodimensional tables in the Firefox or Opera web browser.

When the XML database of Genesis 1:1-2:3 is displayed in the Firefox web browser using the style sheet above, the results look as shown in Figure 4.7 (only the first two clauses are shown; see Addendum K for the whole file).



Gen01v01a

Level	Phrase1	Phrase2	Phrase3	Phrase4	Phrase5
Phon:	bre$it	bara	elohim	et ha$amayim ve'et ha'arets	-
Translation:	in the beginning	he created	God	the heaven and the earth	-
Phrase type:	PP	VP	NP	NP	-
SynF:	Adjunct	Main verb	Subject	Object	-
SemF:	Time	Action	Agent	Product	-

Gen01v02a

Level	Phrase1	Phrase2	Phrase3	Phrase4	Phrase5
Phon:	veha'arets	hayta	tohu vavohu	-	-
Translation:	and the earth	was	an emptiness and void	-	-
Phrase type:	NP	VP	NP	-	-
SynF:	Subject	Copulative verb	Copula-predicate	-	-
SemF:	Zero	State	Classification	-	-

Figure 4.7. The first two clauses of the XML clause cube as displayed in the Firefox web browser as two twodimensional tables.

This presentation of the threedimensional XML clause cube as a series of twodimensional tables, viewed in an internet browser, may be regarded as a simple visualisation of the data. The format and appearance of the file could be changed relatively easy by changing the style sheet.

Although this representation at first sight looks very similar to the representation in Figure 3.17, it is actually very limited. It does allow simple searches using the browser's built-in functionalities, but does not present the required data clause by clause because the formatted data is presented as one long web page. This limitation could become quite problematic in huge data sets. Furthermore, users cannot search the data specifically on clause numbers or verse numbers; neither can they slice off required linguistic modules or expect any new requirements to be fulfilled. The browser interface is, therefore, only suitable for simple uses and cannot facilitate advanced processing of the data. Later chapters (Chapters 5-6) will, therefore, use third generation programming languages to overcome these limitations. Some of the functionalities discussed in Chapter 3, such as slicing and dicing, will be integrated with more advanced procedures. Create, read and update functionalities will be added. Data mining and visualising the XML data in custom-made ways will be utilised to look for interesting patterns in and across the various linguistic modules.

4.13 Conclusion

The empirical exercise in this chapter proved to be quite successful. It showed that XML can be used to build a multidimensional database of linguistic data, which can be visualised as a series of twodimensional tables by using a style sheet and web browser. It showed that a database approach to capture and manipulate linguistic data is a viable venture in computational linguistics and an example of natural language information systems. Various layers of linguistic data were captured in an XML document using the phrase as the basic building block of the data cube. The data may also be imported by a VB6 program for user-friendly viewing or editing purposes and rewritten to XML for storage. This process of round-tripping will be discussed in the next chapter. The integration of data in the data cube also facilitates data exploration (see Chapter 6). More complex visualisations of subsets of the data will be discussed in Chapter 7.