Chapter 1

Introduction

This thesis explores the relationship between novel communication techniques based on the Extensible Markup Language (abbreviated as XML in the sequel) [W3C98b] and the well-established field of database management systems (DBMS). The author will try to show differences and similarities, which eventually result in frugal opportunities. This way, the two areas can co-exist and cross-fertilise each other.

1.1 Why XML?

The impressive growth of the World Wide Web was first initiated by the desire of authors and publishers to make electronic documents available to an international audience – instantly, cheaply and easily. It started out as a novel communication link between humans. The Hypertext Markup Language (HTML, for short) [W3C01b] was at that time used to render documents in a visually appealing way: it followed the traditional lines of structuring documents and offered facilities for defining various types of section headings, paragraphs, enumeration lists etc. as well as directives to change the style and layout of a document.

As the Web grew in complexity and volume, it also assumed the role of a communication link between computer programs – programmers started using it as an information channel for intra-application communication. Over time, applications developed into more complex and distributed networks, and the distinction between what was a single application and what was a heterogeneous collection of related but independent programs became blurred. Not only did it become blurred but it turned out that the communication between the components tended to become more and more difficult as each one developed at its own pace and the maintenance of standard interfaces started being a problem aggravated not only by the topology and idiosyncrasies (firewalls, byte-order, etc.) of the networks that were involved but also because of issues like internationalisation and, most importantly, semantic interoperability. XML [W3C98b] is an attempt to tackle these problems – an attempt that is backed by a large and influential community – and it has the potential to, while maybe not overcome, at least ease many of the issues mentioned in this paragraph and to provide good working solutions.

One of the most prominent and wonderful features of the human brain is its ability to do reliable pattern matching in given a context. For example, the reader of this thesis can easily recognise that the bold line near the top of the page represents the name of the current chapter; the chapter again is divided, e.g., into blocks called paragraphs,
which again contain sentences. Computers, in general, do not have this ability: they cannot separate the structure of a document from its content; XML is a way to help them do it. Just as HTML is used to render texts so that they can be processed by humans, XML renders data structures so that they can processed by computers. And just as text can be read and understood by readers with different backgrounds as long as they share a common interest, XML documents can be used by computer programs from different areas of application as long as they share a common vocabulary.

Of course, computer applications are not as clever as the human brain: the programmer needs to specify exactly what part of a document corresponds to which data type and data structure known to the program. XML and its infrastructure have been designed to do just this: to make information virtually self-describing so that it can be used with only little inside knowledge. This rather simple sounding shift in focus gives rise to a new era in information processing. For the first time, computers are given the necessary background and vocabulary to ‘understand’ each other. Just like literate humans, who know what a document looks like and who can spot it without reading or understanding it, computers are given a basic way of communicating with each other that is independent of the operating system, the localisation, the hardware, the network infrastructure. Consequently, XML has spread like a wildfire through science and into industries ranging from engineering to archaeology.

The body that co-ordinates the standardisation of XML, the World Wide Web Consortium (W3C for short, [W3C01a]), plays an important role in the success story of XML. According to its own words, the ‘World Wide Web Consortium (W3C) develops interoperable technologies (specifications, guidelines, software, and tools) to lead the Web to its full potential as a forum for information, commerce, communication, and collective understanding.’ In particular, it tries to make writing applications easier by specifying standards that build on XML and provide complementary functionality to process XML transparently in the various layers of a complex software system. The most prominent among these are:

**Extensible Markup Language (XML)** is the base technology which specifies a set of rules for putting data structures into a text file. Although XML is text, it is not primarily meant to be read by humans but rather by machines, or standardised XML parsers respectively. Although XML is a new format it is the focal point of a long history: it recalls the structure of Lisp’s S-expressions [Ste90, Tou90] and is a direct successor to SGML [GoI90]. Furthermore XML is license-free, platform-independent and well-supported — in short, a reliable technology with a large supportive community. Sometimes ‘XML’ denotes the whole family of tools and standards defined around the markup language as well.

**XPath** is a query language designed to address parts of XML documents. Originally, it was intended to be used by both standards XSL and XPointer. However, it developed into a query language in its own right, and numerous database vendors have incorporated it into their products (see [Mue00, RFG+01, IBM00] among others).

**XPointer** or, more accurately, the XML Pointer Language, is a query language whose purpose it is to address fragments of the contents of Uniform Resource Identifiers (URIs); a URI is “a compact string of characters for identifying an abstract or physical resource.” [BLFIM98]. These URIs must reference a document of primary type `text` or `application` and of secondary type `text/xml-external-parsed-entity` or, respectively, `application/xml-external-parsed-entity`. 

**Introduction**
1.1 Why XML?

XPointer, which is based on the XML Path Language (XPath), supports addressing into the internal structures of XML documents and external parsed entities. It provides ways to examine a hierarchical document structure and choose between its internal parts based on various properties, such as element types, attribute values, character content, and relative position. In particular, XPointer can specify ranges in documents declaratively.

XLink (XML Linking Language) [DM001] extends the concept of HTML hyperlinks. It allows for bidirectionality of links, links with multiple destinations, links that annotate documents without being physically part of the documents, leaving the document read-only, links that are context-dependent queries, links that span regions of a documents, links that can be defined at various granularity levels. Still, XLink is different from XPath in that it provides a way to embed links into URLs and that it can point to entities that are not nodes or sets of nodes.

XSL, which stands for XML Stylesheet Language, is a language for expressing style-sheets [W3C01c]. This means that it is used to specify how to render an XML document for human use, be it in a browser window, on a printer or even in braille. According to the specification XSL consists of three parts:

- **XSL Transformations** (also known as XSLT) is a language for transforming XML documents.
- **XPath** was already mentioned above and has been proved useful in a number of contexts.
- **XSL Formatting Objects** are a 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.

XQuery is a framework to provide flexible query facilities to extract data from real and virtual documents on the Web. It therefore can be seen as the glue between the World Wide Web and database technology. Its final goal is to ensure that collections of XML documents can be accessed as declaratively as Database Management Systems can be queried with SQL [ISO99].

The fact that implementations for many of the above standards were available early and that the major vendors backed the standards fuelled an enthusiastic response from the Web community. In the long run, XML may have the potential to turn the data that are available on the net into useful pieces of information that can be searched for. Thus, it is hoped that XML will help overcome the difficulties that users still encounter when they search the Internet for information.

The reason why it is notoriously difficult to find the information one needs on the Web today arises largely from the way information is published. HTML, the language used to publish Web sites today, suffers from one drawback as shown in Figure 1.1: it only sketches how data should be laid out in the browser window and not what pieces of information they represent. However, this is not HTML's fault but the absence of an additional layer of abstraction that was not taken into account at the time the Web was started.
XML, on the other hand, does not force the document designer to think about the visual presentation of the document but rather encourages designs along certain semantic guiding principles. Therefore it provides an additional level of abstraction in application design, quite similar to the classic distinction between physical and logical data representation in Relational Database Management Systems (RDBMS). Following the tradition of declarative query languages like SQL [Ame86] that make it possible to delegate physical query execution to a query optimiser and processor and let the user concentrate on the logical formulation of a query by using an abstract interface, XML allows the designer to concentrate on the semantics of the documents while users can choose between self- or predefined views: stylesheet mechanisms, which are simple query and transformation languages, are used to derive a visually attractive representation of the document, whereas a more powerful query language is employed to extract and combine information that, for example, is needed by middleware applications such as search engines which build indexes for Internet or intranet sites. Another important reason for using XML are the internationalisation features: it supports a wide range of character sets and thus is able to adapt to many local conventions.

1.2 How XML?

The trend to use markup languages for data interchange originates in two communities who have been using these languages with completely different purposes: the database community and the document processing or hypertext community. The breakthrough of the World Wide Web has brought the two communities together. The experience of the hypertext community in developing and specifying interrelated sets of documents and the ability of the database community to handle large amounts of data have made it possible to deploy the data-intensive and large Websites that are ubiquitous today.
The database community has been using structured documents in the form of flat files that were used to back-up and bulk-load data management systems since the beginnings of large scale data management. In conjunction with the schema information that is used to structure flat files, they can be regarded as instances of structured documents. Later markup languages added some flexibility which was paid for by accepting the fact that data were not so structured anymore – an incident which gave application designers the possibility to adapt flexibly to new situations: the concept of *semi-structured data* was born. This way it became easier to describe complex data which does not naturally fit the tabular schemas dominant in traditional applications. Some new areas of application were office forms, biological data, bibliographic data, medical records or astronomical data.

The hypertext community on the other hand had been working with structured documents since their very own early days. However, they did not abstract the concept they used in a way that allowed for large-scale deployment and consistency control that was common in the database world. With the advent of the World Wide Web, suddenly the necessity to combine concepts from both communities arose. Users want to browse through large amounts of data such as trade catalogues, bibliographies of not only departments of university libraries but of nation- and continent-wide networks of academic institutions, possibly using integrated views over different databases – just as if there was just one large virtual database. This called for well-defined views over many data sources and XML has become *the* way to specify the views and exchange data.

In some sense, XML would not have been necessary if there had been a single standardised way to exchange data structures. After all, what people want to exchange are not *data* but *data structures*, *i.e.*, data in a context with a meaning. Languages like Lisp [Ste90] have been providing the ability to marshal internal data structures for long and thus rendered the need for a separate data exchange obsolete. However, the balkanisation of the programming languages and networking world brought about the need to agree on a data interchange framework. Since XML is supported by the major players in the industry, it has started to fill the hole and is likely to have a good chance to succeed. The stylesheet and query standards built on XML, however, are true novelties in the sense that a different common standard than XML for exchanging data structures would not render them obsolete.

### 1.3 Where XML?

So XML is a generic format to describe data. Database Management Systems are generic tools to store and query data. It therefore seems natural to use the latter to keep collections of XML documents up-to-date. In their standard textbook on Database Systems, [EN89] list four important characteristics that distinguish the ‘database approach’ from traditional document-centric processing applications:

- Existence of a catalogue or data dictionary
- Program-data independence
- Data abstraction
- Support of multi-user views
It is interesting to compare these database characteristics to the design goals of XML applications. Since XML is a textual format, it abstracts from low-level details like bit-wise number representation and thus provides program-data independence. It also brings about data abstraction by equipping the user with a standard set of data modelling tools, which will be presented in the following section in detail. We have already mentioned XML’s support of multi-user views in the previous subsections: the user can choose between a document layout for on-screen presentation versus one for printers. The existence of a catalogue or data dictionary, however, is a point where differences exist. Document designers can choose whether they want their documents to be self-contained, i.e., without providing a description of the structure of the document, or whether it adheres to a schema: a variety of description languages make it also possible for designers to commit themselves to various degrees of strictness. This open nature of XML, on the one hand, makes up for a very flexible standard because it concedes much freedom to the designer. On the other hand, since document designers are often not the user of the document, they may not anticipate all ways in which documents will be used later and not encode enough information into the document structure. While an excess of information can usually be remedied by defining views over documents, for which standardised ways exist, missing or non-derivable information can break the usefulness of applications. It is therefore important to find a balance between flexibility and strictness.

The usual engineering advantages of DBMS-like transaction control and consistency mechanisms only make the combination more attractive. In a stateless environment like the Web, for which XML has been designed, they are as important as in mission critical applications, where DBMSs are usually deployed.

A fundamental question is where XML technology can be used to simplify the architecture of large software systems. Most likely, XML documents will not be used as internal data structures as there are more intuitive and resource-efficient ways to deal with data. Actually, once XML data have been queried by a query language, for many queries the result is actually quite structured and translates naturally to more specific and typed data structures that can be handled more efficiently but which can also be used to assist in proving the correctness of applications. This is also related to a potential hot spot in application programming: the loose typing of many XML data has to be integrated into the typing systems of the host programming languages. Again, schema languages can help to align the two worlds and enable compilers to make use of type information and provide additional safety and correctness measures.

### 1.4 Conclusion and Overview of Thesis

XML has become a fundamental technology, and large amounts of XML data need to be managed. The database community can respond to the challenge by extending the well-established concepts of relational data models and query processing to meet the demands of the XML community, which is large and made up of various interest groups for whom flexible data interchange is an issue.

The rest of this thesis is concerned with methods to manage XML data inside a database management system. In terms of presentation, the following chapter sets out by giving an overview of XML and related technologies like schema definition and query languages. By looking at the plain standard document, XML is simply a syntactical way of describing how to combine text and markup into structured documents. The many standardisation efforts, however, assume that a document designer makes use
of the markup in a way that its structure reflects a semantic modelling process. Thus, XML's content models, i.e., the tag hierarchies used in documents, in many applications resemble real world entities, which, in turn, makes them an interesting target for query languages. Constraint definition languages like DTDs or XML Schema help to restrict the set of permissible document structures and to get closer to the systematic rigidity of data models that are deployed in relational database management systems; these models equip the user with more control over the semantic integrity of the data used in an application, which then may even be ensured automatically or semi-automatically, than the XML standard does. But the primary reason why it is interesting to look at how XML and database technology can benefit from each other is not only that there are parallels between data-oriented XML documents and the table-based relational data model of DBMS, which suggests that there are a number of areas where both worlds can benefit from each other, but because XML has become the lingua franca on the Internet and appropriate frameworks and tools are being deployed in many applications anyway as pointed out in the following chapter.

The high degree of freedom which document designers enjoy also brings about the downside that only very general semantics can be assigned to the XML elements that are used in document instances, since they are purely syntactic constructs. This becomes apparent when a semantically strict data model like a relational data model is to be used to model and store XML data. Therefore, a characterisation of physical data breakdowns is presented in Chapter 3, and Monet XML, the binary data model which underlies most of the research reported in this thesis, is discussed in detail. Shortly, the implications of the data models on other components of the DBMS-like the transaction manager and the query optimiser are discussed. Experiments show that storing and querying documents in space-efficient, binary Monet XML is interesting since many of the problems inherent in semi-structured data become easier to handle; the prototype implementation underlines the viability of the approach: the effort to reduce data volume quickly pays off as gains in efficiency. Overall, the Monet XML approach combines clear semantics with an efficient execution model by means of a simple and effective mapping between XML documents and a relational schema, which also simplifies the query algebra. However, before documents become available for querying, they first have to be bulk loaded into the storage engine. Chapter 4 of the thesis is concerned with issues around querying documents and presents a bulk load algorithm that exploits the semantics encoded into the document tree-structure to achieve favourable performance. Additionally, the work discusses the trade-off between navigational updates, which are characterised through random access patterns, and bulk updates, which exploit the set-oriented storage of Monet XML.

Then we set out to explore how regular path expressions, the feature that probably most distinguishes XML querying from the standard relational algebra, can be implemented on a relational engine. Our strategy is to use the summary information that Monet XML automatically provides in a pre-processing step and to expand regular path expression to joins by removing wildcards and replacing them with the union of the paths they represent.

The presentation of query primitives is wrapped up by the introduction of the meet operator, which is a tool to allow users without domain knowledge to query XML databases; this is done by exploiting the semantics that are encoded in the hierarchically nested tag structures that are the back-bone of XML's syntax. The key idea of the operator is to find the smallest enclosing context of certain groups of nodes which are typically the results of a full text search; this idea is reduced to a lowest common ancestor search in the document tree.
Since XML data potentially contain many dependencies that escape the constraint mechanisms deployed in state-of-the-art query processors, we also propose to augment the SQL layer of a relational query processor with the Choose operator, which syntactically is a Union-compatible operator. The operator declares that its two inputs are semantically equivalent and leaves it to the query optimiser to choose which one to execute depending on cost estimates. Although the operator is sensible not only in XML settings but in many application areas like Geographical Information Systems or On-Line Analytical Processing, XML can especially benefit from it. The plethora of ways to physically break down data each come with specific advantages, disadvantages and constraints that may be hard to declare to a query processor. The Choose operator can help here to enlarge the optimiser's search space with promising alternatives that otherwise could not be uncovered. A focal feature of the operator is that it seamlessly integrates into the existing optimiser infrastructure.

The research is concluded in Chapter 5 by looking at what determines the performance of XML databases in general: the infrastructure around that database engine may be as important when an application is deployed as the (customised) database product itself. One of the primary reasons for introducing XML was to channel programmers' workflow by making data exchange both machine and programming language independent and, thus, more robust. XML also fosters standardisation by encouraging programmers to make the data structures used in communication public. The goal is now to have the XML database integrate smoothly into this all-XML framework; on the one hand, this is a software engineering challenge, on the other hand, since the database management system usually is the component to cope with large amounts of data, performance is often mission critical. To help users evaluate the trade-offs in this situation, the thesis presents the XMark benchmark. It provides a document database base and a query set to model query scenarios and help users analyse the performance characteristics of their XML systems. While the query processor is challenged with the benchmark, users can also do a requirements check on the set-up and the embedding infrastructure. Eventually, Chapter 6 wraps up the thesis by presenting a summary and some directions for future research.

1.5 Bibliographical Remarks

[ABS99] is a good introduction to Web data in general and XML in particular from a theoretician's point of view with interesting historical remarks. In contrast, [BB99] motivate the necessity for XML in data processing from a layman's perspective and present excellent illustrative examples. In the context of SGML, the question about expressiveness of markup languages have been raised; [Woo95] gives a very readable overview. More detailed discussions on parsing markup languages and validating content models can be found in [BKFW97] and [MGH97].

Relational database systems and documents management systems share common roots in office data management: relation technology takes care of data which comes in tabular form whereas document management systems are used to process form data. Both areas received great attention from the research community. Amongst many other interesting papers, [ACM93] describe how to use object-oriented database technology in the context of structured files. [GZC89] present an algebra for structured office documents, one of whose salient features is that it provides (ordered) tuple sequences instead of plain relations. Historically, [Tsi80] was among the first to report on an implementation of a special purpose office automation system that includes flexible form
processing. [HP01] present an interesting approach to integrating XML documents into the type system of a programming language.