Usability of XML Query Languages

Bruikbaarheid van XML zoektalen

(met een samenvatting in het Nederlands)

Proefschrift

ter verkrijging van de graad van
doctor aan de Universiteit Utrecht
op gezag van de Rector Magnificus, Prof. Dr. W. H. Gispen,
ingevolge het besluit van het College voor Promoties
in het openbaar te verdedigen
op maandag 17 oktober 2005 des ochtends te 10.30 uur

door

Joris Petrus Maria Graaumans

geboren op 20 november 1974 te Eindhoven

promotor: Prof. dr ir G.J. van der Steen,
oud-hoogleraar aan de Universiteit Utrecht
copromotor: Dr H. van Oostendorp,
Instituut voor Informatica en Informatiekunde, Universiteit Utrecht

SIKS Dissertation Series No. 2005-16. The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

Dit proefschrift werd mede mogelijk gemaakt door de SGML/XML Users Group Holland.

ISBN: 90-393-4065-X

Acknowledgements

I would like to thank everyone who contributed to this thesis or who supported me during my Ph.D. period. First of all, my promotor Gert van der Steen and former supervisor Hans Voorbij for providing me with the opportunity to become a Ph.D student. I would like to thank Gert for his many ideas and enthusiasm, the discussions and conversations about research and non-research topics. I greatly appreciate the efforts he took to supervise my work. Especially because he no longer held an official position at the Utrecht University and was supervising me basically in his free time the final year.

I would like to thank Herre van Oostendorp for being my copromotor. He convinced me of the value of experimental research and helped me to design and conduct usability experiments. I think this thesis would be very different (and not better) without his influence. Roelof van Zwol provided me with the experimental platform for my experiments, and made some very useful comments in various stages of this research. More importantly: he is a very nice person who is great in putting things into the correct perspective. Thanks!

I would like to thank the members of my promotion committee, Arno Siebes, Jurgen van den Berg, Gerrit van der Veer, and Geert-Jan Houben for their reading and useful comments.

At this moment I no longer have a job at the Institute of Information and Computing Science of the Utrecht University. It was an excellent place to work and that was not because of the nice building. Thanks to my former colleagues for the pleasant working environment and many thanks to those who were so kind to share rooms with me: Joske Houtkamp, Ion Jovina, Stacy Nagata, and Rik Bos. In addition, I would like to thank Ion for his help with statistics and SPSS.

It is impossible to conduct usability experiments without appropriate subjects. Therefore, I would like to thank everyone who participated in the experiments. That is: the members and the former members of the Large Distributed Databases Group and many of the students Information Sciences.

Special thanks to Gerard Cornelissen, Steven Cornelissen, and Jolanda Graaf who sacrificed a considerable part of their time to proofread this thesis.

I spent quite some time and effort writing this thesis. I am very grateful to my band members who spent time with me in sweaty and noisy rehearsal rooms and studios, and in almost empty and crowded venues. Thanks to Arnold, Edward, Joachim, Jan-Jacob, and Manuel.

I would not be here (or anywhere) without the support and love of my parents Henk and Joke, my brother Harry, and my sister Marloes. I'm also very happy with the fact that my brother and sister are willing to act as paranimf during the defense.

Although I am very proud of and happy with this thesis, related to other things in life it does not mean much at all. I would like to express my deepest gratitude to Isabelle and my two beautiful daughters Anne and Sofie just for being there.

Summary

The eXtensible Markup Language (XML) is a markup language which enables re-use of information. Specific query languages for XML are developed to facilitate this. There are large differences between history, design goal, and syntax of the XML query languages. However, in practice these languages are used for similar purposes. The motivation for a particular design decision is often not clear. Also it is not clear if a particular design choice influences the user's effectiveness with a query language.

In this study the usability of most important query language candidates are investigated. Usability is: the performance and the satisfaction of users measured in relation to a set of tasks. Not much is known about the usability of formal languages. In chapter 2 an overview is provided of former research of the usability of database query languages. The methodology and used models are discussed.

The most important assumption in this study is that differences between the syntax of query languages lead to differences between usability of these languages. In chapter 2 the background of this assumption is discussed. In the XML community there have been fierce debates about the assumed influence of syntax on the usability of a language. A common argument in these discussions is, for instance, that XSLT is difficult to use because of the verboseness of the language. XQuery is argued to be more suitable for users with a database background because of the compact syntax that is partly based on popular database query languages.

Performance differences between users of XML query languages must be related to how users formulate queries with these languages. If a user needs more time or has more difficulties when formulating a query with one language compared to another language, these languages must differ in how queries are solved. In chapter 3 a model is presented how users formulated queries with a query language.

This model is used to interpret behavior of subjects participating in a thinking aloud experiment. This experiment provides a first impression of the usability of three XML query languages (XSLT, XQuery, and SQL/XML) and the possible reasons for these differences. The results indicate that the model is a suitable representation of the query solving process. This experiment is discussed in chapter 4.

The differences between query language usability are investigated further in a second experiment. A large group of subjects and a set of five query tasks show that users perform better and are more satisfied with XQuery compared to XSLT. This experiment is discussed in chapter 5.

A third experiment was conducted to explain the causes of the difference in usability between XSLT and XQuery. In advance the following possible reasons for performance differences are mentioned: the complexity of query tasks, the structure of XML

documents, the verboseness of language expressions, the different methods of embedding Xpath in XSLT and XQuery, user experience, and the number of expressions that are necessary for a particular query task. These factors are quantified for a large set of query tasks and the influence of these factors is determined on the performance with a query language. This experiment is discussed in chapter 6.

This study shows that the usability of XQuery is higher compared to XSLT. XQuery is easier to use because the expressions in this language are more compact and because XQuery embeds Xpath in a more simple way compared to XSLT. Furthermore, query complexity is a good predictor for user performance on a query task. Finally, user experience has a large influence on the performance with a query language. The influence of different XML structures on performance with a query language was not shown in this study. The query tasks and the subjects in the experiments are representative for the start of the learning curve of these languages. The conclusions of this study are mentioned in chapter 7.

Table of Contents

Chapter 1. Introduction

1.1. The Need for a Usable XML Query Language

The amount of data produced, stored, and reused has increased rapidly over the last years. Data is stored so it can be reused later. Reused means: the data is exchanged between applications, or is rendered to make the data understandable for human users. Applications can only process data that conforms to a specific structure. Therefore, data that is exchanged between applications must apply to this structure or the original structure of the data must be adapted to this structure.

Several examples of data exchange between applications are provided by Van der Steen [VdS01]. We mention three of them:

- An ambulance is on its way to the hospital. The patient is identified. The information about the patient is searched with a mobile device in a database. This database consults other databases, for instance the databases of the patient's general practitioner and the insurance company. The data is sent to the specialist in the hospital who also listens to the temporary diagnosis of the patient by phone. One moment later the printer of the ambulance prints an advice for an emergency treatment together with the medication that is required for this patient.
- A mechanic is performing the periodical service to an airplane. He encounters some unexpected problems and searches the appropriate repair instructions with his portable computer. He puts on his goggles with built-in LCD screen so he can use both his hands. The screen shows a figure of the part he needs to repair together with the matching instructions. When a part is broken he can order the part with his portable computer.
- A person logs in to the Internet and enters his or her own webportal that is tailored to that person's interests. The latest news from the world, country, and the local area are presented in categories that are relevant for this person. New articles and special offers that are interesting for the person are listed and can be ordered with one click of the button.

One of the most important data structuring languages for the Internet is XML (eXtensible Markup Language) [XML]. The reason for XML's popularity is that the language is very simple and platform independent. Also, there are a large number of support tools available. XML can be used to encode all kinds of data, for instance the plays of Shakespeare, relational data, and technical manuals. See the following two examples of two different news sources that are structured with XML:

Example 1.1. News.xml: a News Source

```
<news>
<message>
 <topic>Politics</topic>
 <title>Blair secures third term</title>
 <body>
 Tony Blair acknowledges the Iraq effect but
 urges unity as he wins a third term with a
 greatly reduced majority.
 </body>
</message>

<message>
 <topic>Sports</topic>
 <title>Glazer wins control of Man United</title>
 <body>
  US sports tycoon Malcolm Glazer has won
  control of Manchester United in a
  790 million pound takeover bid.
  </body>
</message>

<message>
<topic>Weather</topic>
<title>Tomorrow's weather</title>
<body>
 Dry with sunny spells at first but showers,
 these becoming heavy and thundery.
 Moderate south westerlies.
 Maximum temperature 17 deg Celsius
</body>
</message>
</news>
```

Example 1.2. Newsitems.xml: another News Source

```
<newsitems>
<item>
 <subject>Economics</subject>
 <headline>Dollar hits new record low against Euro</headline>
 <location>Berlin</location>
 <text>
 The U.S. dollar hit an all-time low against the Euro,
 which breached the $1.35 mark after a mixed economic
 report from the U.S. Commerce Department.
 </text>
</item>

<item>
 <subject>Science</subject>
 <headline>Earth's species feel the squeeze</headline>
 <location>London</location>
 <text>
 If we continue with current rates of species extinction,
 we will have no chance of rolling back poverty and
 the lives of all humans will be diminished.
 </text>
</item>
</newsitems>
```

XML documents have a hierarchical and ordered tree structure. The structure is defined by elements. Elements are encoded in documents with tags, or words that are included in '<' and '>' brackets. In the second example "location" is an element.

Let's consider the sample of the webportal. Suppose that a user only wants to see news headers that are related to science and politics in his personal webportal. Different news sources must be integrated: irrelevant news must be filtered and the news must be presented in a format that is understandable by a web browser. The two example documents could be integrated to the following webpage:

Example 1.3. Webportal Page

```
<html>
<head>
    <title>Personal Webportal</title>
</head>
<body>
<h3>Politics</h3>
  <ul>
    <li>Blair secures third term</li>
  </ul>
<h3>Science</h3>
 <ul>
   <li>Earth's species feel the squeeze</li>
  </ul>
</body>
</html>
```

Appropriate query languages are required to enable the exchange of data between applications and to make data usable for human users. Several query languages for XML are proposed by individuals, research groups, and organizations like the World Wide Web Consortium (W3C) and the International Organization for Standardization (ISO). XQuery is the XML query language proposal of the W3C. The webportal page from the previous example could be generated with the following query language expression:

Example 1.4. XQuery expression

```
let $m := doc("news.xml")//message
let $i := doc("newsitems.xml")//item
let $politics :=
 ($m[topic="Politics"]/title/text(),
 $i[subject="Politics"]/headline/text())

let $sciences :=
 ($m[topic="Science"]/title/text(),
 $i[subject="Science"]/headline/text())
return
<html>
<head>
    <title>Personal Webportal</title>
</head>
<body>
<h3>Politics</h3>
<ul>
{
for $p in $politics
return
<li>{$p}</li>
}
</ul>
<h3>Sciences</h3>
<ul>
{
for $s in $sciences
return
<li>{$s}</li>
}
</ul>
</body>
</html>
```

The syntax of XQuery and other query languages are discussed later in this study.

Many XML query languages have a large overlap in purpose and functionality. However, there is still a lot of variation between different query language proposals. The syntax of an XML query language is often motivated with the same argument: "this syntax is well-known by the users and easy to learn". For instance, the requirements of the W3C XML Query Group state that the “XML Query syntax must be convenient for humans to read and write” [W3C-QR]. This statement is of course very general. It is unclear what is meant by 'convenient' and it is not clear what kind of users those 'humans' are. Are these humans information experts, programmers, database experts, or librarians? In any case, the size and complexity of current query languages and the XQuery example above show that querying XML data is entirely different from, for instance, querying the World Wide Web by entering keywords in a search engine. The process of formulating query language expressions to extract

data from XML sources is more comparable to traditional programming tasks and not intended for novice users.

Query languages can be used in two different scenarios. First, a query language can be embedded in an application and in this situation the queries will be generated automatically. Second, a query language is the interface for a user to a data collection. In this case, queries will be formulated by specialized human users, for instance, database experts or information analysts. In this study, we focus on the second scenario and on the usability aspects of XML Query languages. Issues that are related to implementation, like which query language can be implemented most efficiently, are outside the scope of this research. Also, when comparing languages we are limited to the available functionality in current XML query processors. This means that features like updating, full-text search, and the use of XML Schema data types are not within the scope of this study.

Differences between programming languages in the past have lead to fierce debates in user communities. Advocates and opponents of languages sometimes argue over aspects of the syntax of a language with almost religious beliefs. Interestingly, research to the actual influence of syntax aspects of such heavily debated languages is virtually non-existent.

The amount of XML structured data will increase further in the future. The applications where XML needs to be extracted, exchanged, and restructured will therefore also increase. For this reason, it is important that XML query languages are easy to use by the intended user group. Until now, the focus of research and development to XML query language has been on technical and functional aspects of the languages. In this study we will try to understand the effects of syntax aspects on the usability of XML query languages. We want to know which XML query language can be used best for a target audience of users.

1.2. Research Questions

There are several XML query language proposals, but it is unclear how well the intended user group can work with these query languages. Therefore, the main research question is:

- *What is the usability of most important XML Query Language candidates?*

Furthermore, we are interested in the reasons for usability differences. Syntax features are often used in arguments about the ease of use of languages. Also, some languages are claimed to be easier for a certain group of users. In the XML community it is often mentioned that some languages are more suitable for specific types of XML documents. This leads to the following question:

- *What are the reasons for differences in usability between these languages, if any? In particular, what is the influence of the following factors:*
  - *Syntax features of the language;*
  - *Background of the users;*
  - *Structure of the XML documents.*

We attempt to answer these questions by conducting three different experiments. In these experiments, a group of subjects will solve a set of queries with different XML query languages. The performance and satisfaction of the subjects is assessed and used to draw conclusions about the usability of the languages. Each experiment focuses on a different aspect of query language usability and therefore the experiments differ in research goal, setup and, if applicable, statistical methods.

1.3. Outline of the Study

Chapter 2 - Background. This chapter provides an overview of the most central concepts in the study. First, an overview of specific requirements for XML query languages is given, followed by a short overview of Xpath and three candidate query languages: XSLT, XQuery, and SQL/XML. We discuss the origin and most important characteristics of each language. The languages are chosen because they are currently widely used for querying purposes (XSLT), or have a large potential user group (XQuery and SQL/XML). Xpath is discussed separately because XSLT, XQuery, and SQL/XML use Xpath as a language to select nodes in an XML source. The second part of the chapter provides an overview of usability aspects. This consists of a description of usability and its components, and research that has been done to the ease of use of relational query languages. A common research setup is discussed, together with factors that influence the user's performance with query languages. In addition, some preliminary models of the human process of query solving are discussed.

Chapter 3 - Understanding the Query Process. In this chapter we describe our hypothetical model of the query solving process for XML query languages. The sources of the model are discussed, as well as the definition of terms used in the model. In addition, the factors that are found in previous research and that are discussed in Chapter 2 are related to XML Query language. Frequently recurring factors that arise in discussions in the XML community are related to the model of the query solving process. The central question is: what is the influence of these factors on the query solving process? The chapter ends with a short overview of the experiments we conducted to answer the research questions.

Chapter 4 - Experiment 1: Exploring the Model. The goal of the first experiment is to gain more insight in the human process of query formulation. Based on previous research we have formulated a model of how users write queries and we identified possible important factors that influence performance with an XML query language. However, before we can validate and test these ideas, we first want to gather more

qualitative data with a thinking aloud experiment on a small set of subjects. This is done because investigating the usability of XML query languages is a new research topic. Only a few related studies are known and we first want to check whether these findings apply to the usability of the languages included in the study on a small set of subjects. We want to find indications of differences in performance and satisfaction between XQuery, SQL/XML and XSLT, measured with a small set of subjects. In addition, we want to verify our ideas about the reasons that cause performance differences and find other possible factors that influence the usability of XML query languages.

Chapter 5 - Experiment 2: Validation. Goal of this experiment is to verify whether the indications found in the thinking aloud experiment can be validated with a quantitative experiment: Is the usability of XQuery higher than XSLT? Do the user performance differences between XQuery and XSLT become larger on more complex query tasks? Are Xpath embedding, verboseness, and expression type relevant predictors for performance with XQuery and XSLT? The performance of subjects was determined with five representative query tasks.

Chapter 6 - Experiment 3: Explanation. Goal of this experiment is to explain our findings from the previous experiment: why is the usability of XQuery higher compared to XSLT? Another goal is to test whether different document types influence user performance with XQuery and XSLT. Compared to the previous experiment, a larger set of representative query tasks is used and the performance on each task is measured for a small set of subjects. The performance on the tasks is related to the relevant query language properties as represented in every query task.

Chapter 7 - Discussion and Conclusions. In this chapter the results of this study are summarized and discussed. We end this study with overall conclusions, some implications for query language design, and suggestions for future research.

Chapter 2. Background

2.1. XML Query Languages

2.1.1. Introduction

This chapter provides an overview of the most central concepts in the study. First, the global expectations regarding XML query languages are discussed. The W3C organized a Query language workshop [QL'98] in December 1998. The main goal of the workshop was to start the discussion about query languages for the web, to gather requirements for XML query languages, and to gather query language proposals. The workshop was attended by participants from the document, database, and information retrieval communities. After this workshop the W3C XML Query working group was formed and its task is to develop the first world standard for querying Web documents. Important works of this working group include the query language XQuery 1.0 [W3C-QL] and the XML Query Use Cases [W3C-QU], which contains illustrations of applications for an XML query language. In the next section we provide an overview of the most important requirements for XML query languages that were proposed during and after this workshop. The overview of XML query requirements is followed by an overview of Xpath and three important XML query language candidates: XSLT, XQuery, and SQL/XML. These languages are selected because they are widely used or because they have a potential large users group. Xpath is discussed separately, because the three XML query languages include Xpath as a sublanguage. Third, we provide an overview of most important concepts from the field of usability engineering. We are not aware of recent usability studies to programming languages, XML related standards, or XML query languages. Therefore, we considered earlier research to the ease of use of relational query languages in the final sections of this chapter.

2.1.2. Requirements [1]

Requirements for XML query languages are related to the input of a query, the functionality of the language, and the output of a query and the syntax of the language. We provide an overview of the most important requirements for an XML query language.

2.1.2.1. XML Document Types

An XML query language must be able to query both one or more well-formed and valid XML documents. These documents can be real, physical XML files, but also

[1] this section is based on the article "The neglected requirements for XML query languages" [Gra03]

XML views. A query language should handle both types of representations. In addition, it must be possible to query fragments of XML documents.

XML can be used to model information in very different application domains and a query language should be able do deal with these different types of information. The two most important application domains for XML are:

1. *Document-oriented XML applications*: this is the traditional SGML area. An example of a document-oriented application is medium independent publishing.
2. *Data-oriented XML applications*: this area is heavily influenced by the database community. An example of a data-oriented application is data exchange on the Internet.

Document-oriented applications are mainly related to texts that represent natural language utterances. For instance, technical manuals, literary texts, reports, and legal texts. XML documents of this type are characterized by irregular structures, mixed content, i.e. alternations between text and XML elements and one main data type: text.

Data-oriented XML applications are more similar to traditional database structures. This means that data-oriented XML has a regular structure, little or no mixed content, and in addition to textual data type there are multiple other data types that are equally important.

For document applications it is important that the query language can deal with the irregular and untyped nature of textual information. For data applications it is important that the query language can deal with the complex type system of XML Schema [W3C-S2] and that the language can be optimized for use with databases. A universal XML query language should be able to deal with both types of XML applications.

2.1.2.2. Functionality

Most requirements for XML query languages are related to the functionality of the language. The requirements are divided in the following categories: selection, extraction, and restructuring or transformation.

Selecting a document or document part, based on properties of the content and structure, is the most important task for a query language. Content is commonly used to denote the PCDATA of an XML document. Of course, with the advent of W3C's XML Schema also other data types can be used. In Information Retrieval the text is the primary source of information and this community raised most requirements for text operations. The basic method to search in a text or a collection of texts is by means of full-text keyword search. Full-text means that all words in the document are accessible. Keyword searching means that only words (strings separated by whitespace) can be selected. In 2001, the Library of Congress created an extensive list of

requirements related to text searching in XML documents [ABC+01]. Important requirements mentioned here include the use of wildcards when searching in strings and the possibility of proximity search of words, that is finding words within a specific range of other words. Other requirements include that it must be possible to find words in a specific order, use stemming operators, for instance to find all bills containing words stemmed from revoke. Finally, case insensitive searching and the possibility to ignore punctuation marks and diacritical marks.

In 2003, the W3C XML Query Group extended their collection of XQuery related working drafts with two drafts about full-text search, [W3C-FTU] and [W3C-FTR]. However, the first version of XQuery will not incorporate support for full-text search.

In addition to requirements related to the selection and extraction based on the content of XML sources, several requirements have been stated about selection based on the structure of XML sources. An XML document is structured as a labeled tree. Support for regular path expressions can be very useful for searching in unknown structures. The following structural categories are distinguished:

- *Hierarchy:* this includes parent/child relations and ancestor/descendant relations.
- *Sequence and position:* this includes sibling relations and parts of XML sources that can be located before or after other parts of the document. XML documents are ordered like texts and this is in contrast to, for instance, tuples in a relational databases.
- *Links:* can be realized in different ways in XML:
  1. Within a document by means of id, idref and idrefs attributes.
  2. Between documents by means of specific links, for example with a `<lnk>` element.
  3. By means of the Xlink standard.
- *Names:* conditions and operations on the labels (that is: names of XML elements and attributes) must be possible.
- *Datatypes:* The nodes of an XML tree are not only named, they can also be typed. The number of data types that can be used with DTDs is rather limited, but the W3C XML Schema standard provides a large number of data types. A requirement from the W3C XML Query Group is that the query language supports operations on all data types represented by the XML Query Data Model.

In addition to selection and extraction, the restructuring of the result of a query is of great importance in both the database community and the traditional SGML/XML community. This is in contrast to keyword-based search on the web, where the result of a query is often a pointer to the information source. These pointers are ordered according to a relevance ranking mechanism. In the traditional SGML/XML world transformation is important because one or more representations of one source document need to be produced. Also, in the database world the result of a query often needs to be restructured. For example, existing data needs to be summarized in a new table, data from multiple table needs to be combined, or existing data needs to be

updated. Queries must be able to transform XML structures and must be able to create new structures. Besides constructing new structures, the following operations should be possible:

- *Sorting*: sort the result based on a specific condition.
- *Unorder*: transform a sequence to a non-ordered collection.
- *Flattening*: create a flat collection of a collection of collections (of any nesting depth).
- *Grouping*: group information items together.
- *Aggregate functions*: use typical SQL aggregate functions like min, max, sum, count, and avg.
- *Preserve order and association*: preserve the original document order of elements if the order is significant.
- *Reduction*: remove subelements of an element. This is also called filtering and pruning.
- *Combination / Joins*: combine related XML fragments from different parts of a document or from multiple documents.
- *Updates:* adjust the source document, for instance by deleting, inserting, renaming and replacing XML fragments.

In this section we provided an overview of the most important requirements regarding functionality of XML query languages. We continue with requirements about the output of queries and the syntax of XML query languages.

2.1.2.3. Query Output

The result of a query should be a new XML document, a collection of XML documents, a fragment of an XML document or a list of links or pointers to relevant information. The Information Retrieval community adds a few other requirements. These requirements are related to relevance ranking, precision and recall. This means that, from an information retrieval point of view, XML query engines should retrieve the best results possible. If no exact matching documents are found, results similar to the query should be retrieved and ranked according to their similarity.

2.1.2.4. Syntax

Query language designers have different user types and communities in mind, and this resulted in several different syntax proposals:

- *Based on SQL/OQL syntax*: languages with this type of syntax are designed for users with a database background. For example, XML-QL [DFF+99] uses a syntax that combines characteristics from traditional query languages like SQL and OQL, and XML syntax. This form is adopted by Quilt [CRF00] and via Quilt by XQuery.

- *Based on XML syntax, or on an XML related standard*: languages with this type of syntax are designed for users with an XML/SGML background. For instance, XSLT uses XML syntax. An advantage of using XML syntax is that the query language and XML will be mutually embeddable. Another advantage is that queries can be accepted by XML processors. For the selection part of a query language is Xpath the most important candidate. Quilt, XQuery, and XSLT are using the Xpath syntax for addressing parts of within XML documents.
- *Based on a graphical syntax*: languages with a graphical syntax are designed for users that are less familiar with formal, textual languages. With a graphical language, the user formulates queries by creating graphical representations of the XML document and query language expressions. The general idea is that a graphical language is easier for naive users compared to traditional textual languages. Several people proposed a graphical query language for XML. Examples of graphical XML query languages are XML-GL and Xing. However, development and usage of graphical XML query languages is not very intensive and therefore we will not consider graphical languages in this study.

In the following section we describe three important candidate XML Query languages.

2.1.3. Candidates

We consider XQuery, XSLT, and SQL/XML as important XML query language candidates because they satisfy the requirements mentioned in the previous section and because mature implementations are available for these languages. For each language we provide:

1. The history, origins, and current implementations;
2. The most important distinguishing properties;
3. A code example;
4. If applicable, critics about the languages.

We start our description of the three candidate query languages with a brief description of Xpath. Xpath is a language that is developed by the W3C to select nodes in XML sources. Xpath is included by several other languages as a sublanguage for precisely this purpose. The three candidate query languages XSLT, XQuery, and SQL/XML all included Xpath as a selection mechanism. However, Xpath cannot be used as a complete query language, because it lacks important query language functionality. For instance, nodes from an XML document cannot be reordered, restructured or updated with Xpath. Therefore, we do not consider Xpath as a complete XML Query language.

We provide an example query expression for each language that is described. The query task description is as follows.

Example 2.1. Query Task

Select the price of the book with title "XQuery from the experts".

The example XML document is as follows:

```
<bookprices>
 <book>
 <title>Data on the Web</title>
 <source>bstore.example.com</source>
 <price>34</price>
 </book>
 <book>
 <title>XQuery from the experts</title>
 <source>bstore.example.com</source>
 <price>55</price>
 </book>
 <book>
 <title>XSLT: programmers reference</title>
 <source>bstore2.example.com</source>
  <price>35</price>
 </book>
</bookprices>
```

The different solutions for this query task are discussed for Xpath and for each query language candidate.

2.1.3.1. Xpath

Xpath (XML Path language) is a W3C recommendation for selecting and addressing nodes in an XML document [Xpath]. Initially, Xpath started as a subset of the eXtensible Stylesheet Language (XSL) but became an independent language in an early stage of the development of XSL. According to the homepage of The Extensible Stylesheet Language Family [XSL]: XSL is a family of recommendations for defining XML document transformation and presentation. It consists of three parts:

1. XSL Transformations (XSLT): a language for transforming XML. XSLT is discussed in the next section.
2. the XML Path Language (XPath): an expression language used by XSLT to access or refer to parts of an XML document.
3. XSL Formatting Objects (XSL-FO): an XML vocabulary for specifying formatting semantics.

Xpath version 1.0 became a W3C recommendation in November 1999 and is used in XSLT, Xlink, and Xpointer. In addition, several query languages used the Xpath syntax for the selection of XML nodes, e.g., XQL, Quilt, XQuery, and SQL/XML.

Xpath 2.0 is currently developed to overcome some limitations in Xpath 1.0: it is a much more powerful language. Xpath 2.0 is developed jointly by both the XQuery Working Group and the XSLT 2.0 Working Group. Some important changes compared to version 1.0 are the support for XML Schema Data Types, the addition of several new functions and operators, and the possibility to use variables.

The following example shows an Xpath expression that is a correct solution of Example 2.1, “Query Task”:

Example 2.2. Xpath Expression

```
/bookprices/book[title = "XQuery from the experts"]/price
```

Without going into details: this expression selects from an XML document with a top level element 'bookprices', all 'price' elements that are child elements of 'book' elements that have a 'title' element with value "XQuery from the experts". The result of this expression is:

```
<price>55</price>
```

2.1.3.2. XSLT

XSLT (eXtensible Stylesheet Language - Transformations) is another member of the Extensible Stylesheet Language family. This family is based on the SGML processing standard DSSSL, which is very complex to implement: no complete implementation of the DSSSL standard has been made till now. An indication that XSLT is easier to implement, is that at this moment 26 XSLT implementations exist, including XSLT support in MS Internet Explorer and Mozilla / Netscape.

XSLT is a language for transforming XML structures in one or more other XML structures. Originally, XSLT was intended to transform arbitrary XML structures to an XSL-FO document but nowadays XSLT is also used to transform XML to (X)HTML, to other XML structures, and to extract fragments from XML documents. Transformations are described in programs that are called stylesheets or transformation sheets. Kay mentions the following important characteristics for XSLT [Kay04]:

1. XML-based syntax: XSLT transformation sheets are XML documents themselves. XSLT is an XML vocabulary.
2. Declarative, functional programming model: the basic programming paradigm of XSLT is functional programming.
3. Rule-based: An XSLT stylesheet is expressed as a collection of rules, in the tradition of text-processing languages like *awk* and *sed*. The rule consists of a

pattern (an Xpath expression) to be matched in the input, and instructions for generating nodes in a result tree when the pattern is matched.

4. Two-language model: XSLT uses Xpath as a sublanguage to select data from the source document.

The following example shows an XSLT solution for the query *Select the price of the book with title "XQuery from the experts"*:

Example 2.3. XSLT Expression

```
<xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="2.0">
<xsl:template match="text()" />

<xsl:template match="/">
<result_query>
<xsl:apply-templates />
</result_query>
</xsl:template>

<xsl:template match="bookprices
 /book[title = 'XQuery from the experts']">
    <xsl:copy-of select="price" />
</xsl:template>
</xsl:transform>
```

This XSLT sheet is an XML document with the element `xsl:transform` as the document element. In XSLT, namespaces are used to distinguish between XSLT elements and literal result elements. In this example, `xsl:transform`, `xsl:template`, `xsl:apply-templates`, and `xsl:copy-of` are XSLT elements. The element `result_query` is a literal result element, this means that the element result_query is constructed in the result tree if the template rule in which they occur are matched. We now briefly discuss the template rules that construct output in the result tree. The second template rule in this example matches the document root (`xsl:template match="/"`). When the root is matched, an element `result_query` is created in the result tree. The `xsl:apply-templates` element is an instruction that selects further nodes from the source tree that are processed when templates in the sheet match these nodes. If the `xsl:apply-template` element would be missing here, the result XML tree would be an empty `result_query` element. In this example, the third template rule matches `book` elements that are child elements of `bookprices` elements and have a `title` child element with value "XQuery from the experts". The `copy-of` element copies the element `price` to the result tree.

The first template rule in this example is added to prevent unwanted output, but is not further discussed here. The result of this XSLT sheet when applied to the example XML document is:

```
<result_query>
 <price>55</price>
</result_query>
```

XSLT is very popular, but it is also criticized for being difficult to learn for users of more traditional procedural languages because of the declarative, functional nature. Another point of criticism is the verbose nature of XSLT that is due to the XML syntax of the language.

A new version of XSLT is under development at the W3C: XSLT 2.0. This version supports XML Schema, Xpath 2.0 and solves some user inconveniences that were noticed in XSLT 1.0. For instance, facilities for grouping problems were simplified and constructions to output multiple XML documents were added.

2.1.3.3. XQuery

XQuery originates from the database community and is intended to be the 'SQL for XML'. The designers of XQuery were inspired by SQL and several query languages for semi-structured query languages like XML-QL, YaTL, XQL and Quilt. The core of the XQuery syntax is the FLWOR expression: for-let-where-order-return that is roughly similar to the select-from-where clauses in SQL. XQuery is defined as an extension of Xpath 2.0 and is also heavily tied to the XML Schema specification. XQuery is mainly intended to be used in combination with an XML database. Several partial implementations and prototypes exist but until now only a few support the XML schema related features. Major database vendors have announced to include XQuery in their products.

The following example shows the XQuery solution for the query *Select the price of the book with title "XQuery from the experts"*:

Example 2.4. XQuery Expression

```
for $b in /bookprices/book
where $b/title = "XQuery from the experts"
return
   <result_query>
    { $b/price }
   </result_query>
```

An important distinguishing characteristic of XQuery are the SQL-like FLOWR expressions. In the example above the `for-where-return` clauses are used. The `for` and `let` clauses bind variables to certain values. In this example, the `for` clause binds the result of the Xpath expression `/bookprices/book` to the variable $b. In the `where` clause, the result of the Xpath expression is filtered: in this example, only values of variable $b that have a `title` child element with value "XQuery from the experts" are selected. The `return` clause determines the result of the query expression. The `return` clause in the example consists of an element constructor, namely the element `result` that contains expressions that select specific values from the source XML. In this case: the value of the `price` element of the nodes that are bound to variable $b. Note that curly brackets are used to switch between XML element constructors and XQuery syntax. The result of this query is the same as the result of the previous XSLT example:

```
<result_query>
 <price>55</price>
</result_query>
```

An often-recurring discussion item about XQuery, is that according to XSLT advocates XQuery has a very large functional overlap with XSLT, but lacks some very convenient features from XSLT. See for instance a summary of a discussion on XML-DEV from 2001 [Smi01] and, more recently, the discussion topic on XML-Dev in December 2004 entitled *What niche is XQuery targeting?* [XML-DEV]. XQuery advocates claim that although the functional overlap is indeed very large, XQuery is more suitable for optimization and therefore more suitable for large XML collections that are stored in databases. Advocates also claim that XQuery is easier to learn and use because the language is more similar to regular programming languages and is less verbose compared to XSLT. On the other hand, XSLT advocates claim that XQuery is mainly suitable for relative simple query tasks and document types, in particular for data-oriented XML documents. The discussion about the ease of use of the syntax is not over yet. One goal of this study is to find a relationship between syntax properties and usability of a language.

2.1.3.4. SQL/XML

In addition to these W3C directed languages, another query language for XML is under development: SQL/XML [SQL/XML]. This is an extension of the ANSI/ISO SQL standard to support XML data that is stored in a relational database. SQL/XML defines specific conversion functions so that XML data can be used in combination with the select-from-where clauses that operate on relational data. XML data is selected with Xpath expressions. The standard is still under development, but three major database vendors have implemented SQL/XML, in combination with vendor-specific extensions. See the following solution for the query *Select the price of the book with title "XQuery from the experts"* below. The example XML document is stored in a relational database where each `book` element is stored in a separate row. The root element `bookprices` is therefore not used in this example.

Example 2.5. SQL/XML Expression

```
SELECT XMLELEMENT ("result_query", extract(e.content,'/book/price'))
FROM example e
WHERE extract(e.content, '/book[title = "XQuery from the experts"]');
```

The first extension function in this example is XMLElement(). This function is used to construct XML instances and needs at least two arguments: an element name and the content of the element. The name of the constructed element is `result_query`, and the content is defined with the second extension function, namely extract(). This function can be used to retrieve XML fragments. In this example, the first extract function has two arguments:

1. `e.content`, this is the name of column in the table where the XML data is stored. Note that the `e` refers to the table alias `e` in the `FROM` clause.
2. `/book/price`, this is a string that contains an Xpath expression. Note that the expression must be delimited by apostrophes. The Xpath expression selects all `price` element children of `book` elements.

The `WHERE` clause also includes an `extract()` function that filters the `book` elements that have a `title` with value "XQuery from the experts".

SQL/XML is still very early in its development. Up till now, we are not aware of any critics to this language. However, we can make an informal comparison with XQuery and XSLT and consider the same factors as mentioned in discussions about these languages. First, SQL/XML is an extension of an existing query language. Instead of working on one data model, like SQL or XQuery, SQL/XML includes two data models: relational data and XML data. Xpath embedding and XML element constructors are

implemented in SQL/XML with extension functions. This makes the language more verbose compared to XQuery, but SQL/XML expressions are less verbose compared to equivalent XSLT expressions. SQL/XML is intended to query XML fragments that are stored in relational databases and to construct XML data from relational databases. Therefore, the functionality for querying and transforming XML fragments is smaller compared to XSLT and XQuery. In our opinion, SQL/XML has some inconsistent features. First, similar function names have different naming conventions. For example, consider the following function names:

- extractvalue()
- value()
- getstringval()
- getnumberval()

Although these functions share the aspect of returning a value, they have two different suffixes: value() and val(). Second, some SQL/XML functions have only slightly different functionality. For instance, the function extractvalue() is defined as a shortcut function for extract().getstringval(). However, extractvalue() can only be used when the result of the Xpath argument is one node while this restriction does not apply to extract().getstringval(). Therefore, we consider SQL/XML not as consistent as XQuery and XSLT.

2.1.3.5. Summary

The following table informally summarizes the similarities and differences between the query language candidates discussed in the previous sections.

Table 2.1. Feature Overview

Feature	XSLT	XQuery	SQL/XML
Verbose / compact	Verbose	Compact	Intermediate
Consistent	Yes	Yes	No
Programming model	Declarative	Procedural	Procedural
Data model	XML	XML	Both Relational and XML
Intended use	XML documents	XML databases	Relational databases with XML support
Sublanguage	Xpath	Xpath	Xpath

2.1.4. Scope of this Study

SQL/XML, XQuery and XSLT cover a substantial part of the wanted features but there are a number of differences between the query languages and their implementations. For instance, current XSLT processors are slower than XQuery or SQL/XML enabled databases. In a strict sense, this is not a language but an implementation issue. One of the most important arguments for the development of XQuery is that this language is easier to optimize for database applications than XSLT. Development of SQL/XML is initiated because of the popularity of relational databases and the need to store and process XML in a relational database. XSLT is not intended as an XML Query language, but it is used as a query language because an appropriate alternative was not available for a long time.

Current query language implementations are limited in required functionality. First, few query processors support the data types defined in XML Schema yet. Second, full-text search capabilities are still a weak spot in most query language proposals. This issue will probably be solved in future versions of XQuery, Xpath, and XSLT. Some vendors already added full-text search extensions in their implementations. Third, both XSLT and XQuery do not support updating, but for XQuery this functionality will be added in future versions. Some vendors have implemented ad hoc update facilities. SQL/XML is an extension of SQL and is therefore capable of updating. There are some proposals for graphical query language, but up till now the development and usage of this type of languages is rare.

For some requirements, in particular the requirements related to syntax and use of the language, it is not clear how they apply to a specific XML query language. The XQuery syntax has been developed to be easy to learn for people with a database background. However, it is not clear which language is easier to use: XQuery or SQL/XML. It is also not clear whether XSLT can be a more easy alternative for people who are lacking a database background. The question remains which XML query language is best suitable for a particular domain or application. SQL/XML, for example, is restricted to XML data that is stored in a relational database and some XML structures cannot be efficiently stored in this type of databases. XSLT, on the other hand, is widely accepted as a transformation language for XML and a large number of the XQuery use cases can be solved with XSLT. If the speed of query evaluation is not a crucial factor, XSLT can be used for querying tasks.

In this study we consider a query language as an interface for specialized users to an XML source. We focus on usability aspects of these query languages. In the next section we discuss aspects of usability that are related to XML query languages.

2.2. Usability Aspects

In the following sections we discuss the general principles of usability engineering and earlier relevant studies to the ease of use of query languages.

2.2.1. What is Usability?

According to ISO 9241-11: Guidance on Usability (1998), usability is defined as "the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use." The most important components of usability according to this standard are:

1. *Effectiveness in use:* the accuracy and completeness through which users achieve certain results.
2. *Efficiency in use:* the resources utilized in relation to accuracy and completeness through which users achieve certain results.
3. *Satisfaction in use:* the freedom from inconveniences and positive attitude towards the use of the product.

In addition, the concepts of usability and its related components are described in *Usability Engineering* which is a widely used textbook [Nie93]. This book is aimed at the development and evaluation of aspects of a system with which a human might interact. We believe that the concepts of usability defined here also hold for formal query languages because these languages are an interface for humans to access data collections.

Usability is the question of how well users can use the functionality of a system. It is considered to be a subset of the larger concept usefulness, that is the issue if a system can be used to achieve a desired goal. The other component of usefulness is utility or machine usability. This is the question of whether the functionality of the system can do what is needed in principle.

Usability is not an intrinsic property of an interface to a system. It is a concept that is relative to a certain context, or as Nielsen states: “Usability is measured relative to certain tasks and certain users. It could well be the case that the same system would be measured as having different usability characteristics if used by different users for different tasks.” [Nie93]. Nielsen distinguishes the following components of usability:

1. Learnability: how difficult is it to learn a language? To know a language means that a user can use the language with a certain degree of efficiency. Users who start using a system or a language need to learn it. Therefore, Nielsen states that learnability is one of the fundamental components of usability. Although some systems are complicated and are built to perform complicated tasks, in general, ease of learning is an important design goal. Easy to learn means that an average

user can reach a high productivity within a short learning time. Hard to learn systems take more time to reach a certain level of productivity. In other words, how steep is the learning curve of a language?
2. Efficiency: how productive is a user once he has learned the language? This applies to expert users who have learned to use the system. Note that the steady-state performance of an expert user still may be insufficient for the user's goals.
3. Memorability: how difficult is it to remember the language when a user does not use the language for some time? This applies mainly to casual users who have learned to use the system and need to remember how to use it.
4. Errors: what is the error rate for a certain language and how difficult is it for a user to fix the errors that have been made? Nielsen defines an error as “any action that does not accomplish the desired goal, and the error rate of the system is measured by counting the number of such actions made by the users while performing a number of specified tasks ” [Nie93].
5. Satisfaction: how satisfied are the users with a certain language? How pleasant is it to use the system?

The components mentioned above can be understood in terms of the ISO 9241-11 standard that is mentioned above. The error rate is an indicator for the effectiveness and efficiency of a system. The lower the error rate, the more productive a user is. Nielsen's notions of learnability and memorability both can be understood as resources that need to be utilized to complete certain tasks. Therefore, we consider learnability and memorability as components of efficiency.

Summarizing, the usability of a system is dependent on the context in which the system is used. It is relative to the functionality of the system, the users, and the tasks they perform. Furthermore, usability is divided in the following three main components: effectiveness, efficiency, and satisfaction.

It is important to measure all three components when evaluating the usability of a system, because these components are not necessarily correlated with each other. Frøkjær et al. [FHH00] studied the relations between the three main components of usability with an experiment where subjects solved information retrieval tasks and with a literature study that included 19 Human-Computer Interaction (HCI) studies that were published between 1997 and 1999. In their own information retrieval task experiment, all three components effectiveness, efficiency, and satisfaction were considered. Effectiveness was measured as the quality of the solutions, which was assessed by one of the authors and expressed on a five-point scale. Efficiency was measured as the task completion time, and extracted from the subject's log files. Satisfaction was measured as the subject's preference for one of the five retrieval methods that were included in the experiment. Frøkjær et al. found that in their own experiment the correlation between effectiveness and efficiency for different tasks are weak or non-existent. Also, the subjects were in general most satisfied with one particular retrieval method, but the effectiveness and efficiency with this method was

not the highest. This means that users are not necessarily most satisfied about the most effective and efficient system.

From the analysis of the data from 19 HCI studies and their own experiment they concluded that in particular for complex tasks the correlation between effectiveness, efficiency, and satisfaction is weak. They also noticed that several studies included only one or two components of usability, but make claims concerning overall usability. Frøkjær et al. suggest that usability studies should measure all three aspects. In addition, identifying the most critical aspect of usability in a particular situation should be a central part of the usability evaluation. For instance, for routine tasks with stable high-quality results, the efficiency is more important than the effectiveness. For non-routine, or complex tasks, that have less predictable results the effectiveness may be more important than the efficiency.

The most common method to test usability is to conduct experiments. In a typical usability experiment a sample of subjects carry out a number of specific tasks on one or more systems that need to be evaluated. The subjects have to be a representative sample of the intended real users of the system and the tasks used in the experiment should be modeled on realistic use cases.

2.2.2. Usability Testing of Query Languages

Research and development of XML query languages has mainly focused on the functionality of the languages and the performance of query processors. Little is known about the usability of XML Query languages. No research has been carried out on this subject until now. In this section we provide an overview about usability of query languages in general and discuss relevant information from previous research.

2.2.2.1. Usability of Relational Query Languages

During the development of SQL a number of experiments have been conducted to evaluate and compare several aspects of this language with other similar query languages. The general approach in these experiments is heavily influenced by an early study in 1975 called *Human factors evaluation of two database query languages - Square and Sequel* by Reisner, Boyle and Chamberlin [RBC75]. This is one of the first human factors experiments for database query languages. This study has the following three research goals:

1. Can the languages Sequel (which is an early version of SQL) and Square be learned by the intended population of programmers and professional non-programmers?
2. Is there a difference in the ease of use between the two languages?

3. Discover types of user errors that were commonly made.

To answer these questions, a common teaching curriculum was developed and used to teach the two database query languages to four groups of subjects. For all four groups, the materials used were as similar as possible and several example databases were used in teaching to accustom the students to different databases. The subjects were taught in regular classes with a highly interactive character. The pace of each class was determined by the slower members. The performance of subjects who were writing database queries was measured both during the classes and after the two-week class sessions in a final exam.

The database query languages, Sequel and Square, are both relational database query languages. Reisner classifies the languages as follows: Sequel syntax is based on English keywords, while the Square syntax is based on mathematical notation. The central concept in both languages is that of a "mapping", which returns the data-values in a number of columns which are associated with a known data-value in another column. The result of one mapping can be used in another mapping: this is called composition. Both languages allow built-in functions AVG, SUM, COUNT, MAX and MIN. For complex queries like grouping queries, the features of the two languages differ slightly. Square uses 'free variables', Sequel avoids free variables and uses the GROUP BY function instead. Both languages use set-operations UNION, INTERSECTION and DIFFERENCE. Both can perform joins. Insertion, deletion and updates can be expressed, but were not taught and tested.

The subjects consisted of 61 undergraduates and 3 graduate students from the local university. The subjects were not preselected and varied considerably in background. The students were divided in two groups, namely programmers and non-programmers. Anyone who took at least one programming course was a programmer and the rest was a non-programmer. The following four groups and classes were made:

- Group 1, Sequel for programmers (18 subjects)
- Group 2, Square for programmers (11 subjects)
- Group 3, Sequel for non-programmers (15 subjects)
- Group 4, Square for non-programmers (20 subjects)

The test was conducted as follows. The subjects were given five tests during the series of classes, and they engaged in a final exam. A memory test was given one week later. The tests consisted of a set of questions which were presented in English, for instance: *Find the names of employees in Department 50*. The student was required to write the appropriate query language statement. The English questions on the tests were the same for all classes. 35 of the 40 questions on the final exam were 'basic feature' questions like mapping, composition, free variables, and set operations. The five remaining questions required subjects to combine some basic features. The subjects used databases they were not familiar with.

A scoring system was developed to rank the subject's responses from less serious error to serious error. This system is discussed in more detail in Section 2.2.2.7, “Query Formulation Process”. Reisner noticed the following. In general, programmers performed better than non-programmers on both languages. Furthermore, non-programmers performed better with Sequel than with Square. Some indications are found about the performance on different features of the languages. For instance, the Sequel GROUP BY function was somewhat easier to use than the free variables in Square, but both features proved to be difficult to use for non-programmers. Invalid syntax errors occurred more often with Square than with Sequel, and more often among non-programmers than non-programmers. A common tendency among subjects was to insert words from the English sentence into the Sequel and Square query, instead of the correct table name, column name or data value. For example, subjects wrote "NAMES" instead of "NAME". All subjects made similar errors.

Some observations about this experiment. First, the research goal is to compare ease of use of two query languages. The method that is used is to teach a language to new users and test their performance on a set of query tasks. To determine the correctness of the result queries a classification of errors was made. Although the researchers are primarily interested in the overall ease of use of the languages, some informal notions about specific features are made based on an error analysis. Some errors are query languages independent, like misspelt column names that are related to knowledge about the database structure. Another query language independent error is when a subject returns the surname only when the full name is asked in the question. This type of error has to do with the interpretation of the question or with sloppiness. Errors related to the query language are, for instance, using a GROUP BY operator for the wrong type of query. This error analysis, and other error analyses that are discussed in the remainder of this chapter, will be used to create a model of the query process in the following chapter.

Second, in this experiment a distinction is made between programmers and non-programmers. Note that on the dimension of computer experience, in 1975 the differences between programmers and non-programmers were probably much bigger than nowadays. Third, the query writing tasks were fairly simple and the database application that was used was very simple. These tasks will probably suffice for evaluating the ease of learning, but we doubt that these tasks reflect real practical uses. Fourth, testing was done by means of pencil and paper. Subjects had to write their queries on a sheet of paper because no online implementation was available at that time.

In the following sections we discuss factors identified in earlier studies that influence the usability of a query language.

2.2.2.2. Query Languages

The main research goal in the previous experiment was to compare the learnability and the ease of use of two query languages. It inspired a number of researchers to conduct similar experiments to the ease of use of other query languages and to specific aspects of query languages. Thomas and Gould, for example, evaluated the ease of learning of *Query By Example* (QBE), a graphical query language for relational data [TG75]. However, due to the lack of control groups and the lack of comparison with other query languages, the only valid conclusion of this experiment was that QBE could indeed be learned to a certain level by a group of students.

Greenblatt and Waxman [GW78] compared three query languages: *QBE*, *Sequel* and a third algebraic query language: *AL*. Their goal was to determine if any of the three languages chosen had significantly better learning and application capabilities over the others when applied to a moderately non-programming sample of subjects. An additional research goal was to verify results of Thomas and Gould [TG75]. The variables of time, accuracy and user confidence as related to question complexity were measured. Besides comparing the languages as a whole, they were also interested in the features of the languages that caused these differences. According to the authors, the languages QBE, Sequel and AL differ in critical aspects such as English - likeness and the declarative vs. procedural specification. However, these differences are not made explicit nor are the notions of declarative vs. procedural clearly defined. This experiment is therefore very similar to Reisner et al. [RBC75], only with different query languages.

A more serious attempt to compare language features is made by Welty and Stemple [WS81]. They assume that there is generally a point in the complexity of any class of problems beyond which procedural specification is simpler than nonprocedural. Their hypothesis is therefore: people more often write difficult queries correctly using a procedural query language than they do using a nonprocedural language. Welty and Stemple used the following classification scheme for relational queries:

- Easy queries cover simple mapping, possibly with arithmetic operations and/or other built-in functions, and composition (chaining).
- Hard queries cover GROUP BY, set functions, joins, and combinations of queries.

Two relational query languages that differ mainly in procedurality are compared, namely *SQL* and *TABLET*. SQL is less procedural than TABLET. In general, of two specifications for the same computation, the one containing more operations and variable bindings that are more strictly ordered, is the more procedural. A metric for this measurement is given, but the terms that are used in this metric mainly apply to SQL and TABLET, and are difficult to translate to other query language paradigms like QBE and XSLT. Two, basically the same, experiments were conducted to test the hypothesis. The second experiment used a slightly different TABLET and the subjects in this experiment had no programming experience. The results of both

experiments confirmed the hypothesis: subjects performed better with the more procedural TABLET on more difficult queries. For the easier queries no differences were found. The differences between the experiments show the following. First, the more procedural language is easier to learn for students without prior computer experience. Second, exposure to languages designed for expressions of procedures (Basic and Fortran) gave the students experience which helped them retain Tablet but not SQL. This is interesting, because it demonstrates that more experienced users not always benefit from additional experience. Third, the performance differences between query languages were only noticeable in difficult queries.

Most researchers classified the languages they investigated in an informal, broad and intuitive manner. The classification is based on the syntax form of the languages. For example, Sequel/SQL is classified as having a textual or linear syntax in [BBE83], [BB82] and [CS95]. Other researchers classify SQL as having a syntax that is based on English keywords, for example in [RBC75] and [GW78]. In addition, SQL syntax is more procedural compared with QBE by [GW78] and compared with TABLET by [WS81].

Two descriptions of procedurality are given in the literature about usability of query languages. First, in an overview article of Reisner [Rei88], the difference between procedurality and nonprocedurality is clarified as follows: a nonprocedural query 'states merely what the result of a query is, not how to obtain it'. For instance, a nonprocedural description would be: "select all names that begin with an 'A'". An example of a similar procedural description would be: "view the first name and check whether it starts with an 'A'. Select the name if this is true. Continue to the next name and do the same.". Reisner notes that the difference between procedural and nonprocedural is not clearly defined. Second, Welty determines the level of procedurality of a statement as follows: for two specifications for the same computation, the one containing more operations and variable bindings that are more strictly ordered, is the more procedural [WS81].

In contrast to SQL, QBE is classified by terms like less procedural and less English-like compared to SQL in [GW78], having a graphical syntax instead of a textual in [BB82] and [BBE83], and having a 'graphical-diagrammic' syntax in contrast to the graphical-iconic syntax of QBI in [CS95].

In these experiments, only one property of a language is used to qualify the language. We think that the qualifications used to denote query languages are too general to explain the performance differences between languages correctly. In the next section we discuss another factor that influences the performance with a query language: user experience.

2.2.2.3. User Experience

Originally, query languages were special purpose programming languages and their intended use was for non-programmers. Most query languages were therefore simpler than normal programming languages. Early human factor experimenters, like Reisner et al. [RBC75] and Welty and Stemple [WS81] focused mainly on the differences between users without programming experience and users with programming experience. The latter are considered expert users and the first are novice users. It is remarkable that users who took one or two programming courses are considered to be expert users.

Nielsen mentions the following user characteristics that can influence usability of a system: age, gender, spatial memory, reasoning abilities and preferred learning style [Nie93]. Some of these characteristics are investigated in the context of database querying. Greene et al. [GGD86] examined the question whether the user's cognitive skills correlate with task performance. The following items of cognitive skills were assessed: reasoning abilities, reading skill, visual/spatial memory, and the ability to integrate complex conditions. The age and years of education of the subjects were also recorded. The following query operator types were used in the test tasks: AND, NEGATION, OR, and AND/OR combined and four different language conditions were used. Two conditions were related to the use of SQL, namely choosing the correct SQL query (SQL Choose) and writing the correct SQL queries (SQL generation). The other two conditions were related to two experimental query interfaces where subjects could enter data constraints in an example table. Results indicate that the user's cognitive skills that are mentioned above influence some aspects of performance in a positive way. Age, however, was correlated negatively with performance.

According to Nielsen, one of the dimensions on which user's experience differ is general computer experience. The more experience a user has, the easier it is to learn and to use new systems. This assumption is not necessarily true. Catarci and Santucci [CS95] conducted an experiment to test whether QBE or SQL is most usable. They distinguished between

1. Naive users, i.e. persons having little or no computer experience
2. Intermediate users, i.e. persons having a general knowledge or computer science, but naive about databases.
3. Expert users, i.e. persons having precise knowledge about databases

They found that naive users perform better with SQL than intermediate users. A possible explanation is that this difference is due to the fact that the intermediate users were mainly used to some types of procedural languages. In this case, it was difficult for them to learn a declarative language such as SQL. Moreover, intermediate users felt themselves confident in solving easy questions, as a consequence they did not put enough effort in solving the easy queries. In contrast, the naive users were completely

unaware of programming techniques, so their mind was free of influences when learning a certain kind of language.

2.2.2.4. Query Tasks

In a typical usability experiment a sample of subjects carries out a number of specific tasks with one or more systems that needs to be evaluated. The performance of the users performing tasks is measured. In this section we give an overview of the different types of tasks. The examples we provide are related on the following example relational database:

Table 2.2. Example Database: EMP

NAME	DEPTNO	SAL	MGR
SMITH	50	12500	EVIAN
JONES	50	11250	MANDLEN
DOE	55	16000	SMITH
ROBERTS	55	15500	JANIS

Table 2.3. Example Database: DEPT

DEPTNO	DEPTNAME	LOCATION
50	STAMPING	SACRAMENTO
55	MILLING	STOCKTON

In an overview chapter about query language experiments, Reisner mentions the types of tasks involved in this kind of research [Rei88]:

- *Query writing task:* this is the most common type of task. The test user receives a question stated in natural language and a (printed) database. The user is asked to write the appropriate database query. One such example is *Find the names of employees in department 50*. The correct response from the user would be the following SQL query:

```
SELECT NAME
FROM EMP
WHERE DEPTNO = 50
```

- *Query reading:* The user receives a query statement and is asked to write the appropriate question in natural language. The purpose of these kind of tasks is to

test whether the user understands query statements. An example of a task like this is:

```
SELECT NAME
FROM EMP
WHERE DEPTNO = 50
```

The correct user response would be: Find/Select/Retrieve the names of employees in department 50. A better method to test whether a user understands a query is the following type of task.

- *Query interpretation:* The subjects receive a query statement and a (printed) database and are asked to find the data asked for by the query. The subjects play the role of query processor. The purpose of these kind of tasks is to test whether the user understands query statements. This understanding is not related to possible language or formulation problems of the user. The query could be the same as the previous example, but the user response would be:

```
SMITH
JONES
```

Query interpretation is in our opinion not the most important aspect of query language use. Most users will be producing queries based on a certain information need. The following task is related to this issue.

- *Question comprehension:* The user receives a question stated in natural language and a (printed) database and is asked to find the data asked for. The purpose of this task is to test whether the user is able to understand the request for information. If the user does not understand the question, it will be very difficult for to produce the correct query. An example of such a query task is: *Find the names of employees in department 50.* The correct user response is:

```
SMITH
JONES
```

- *Memorization:* The user receives a (printed) database and is asked to memorize and reproduce the database. The purpose of this task is to test how difficult the database is for users. In our opinion this is not directly related to practical query language use.
- *Problem solving:* The user receives a problem description and a database. They are asked to generate questions in natural language that would solve the problem. Again, in our opinion this is not directly related to practical query language use.

Most human factor experiments of query languages use query writing tasks to measure user performance because these tasks are similar to the practical use of a query language. Query writing tasks consist of two components. First, the user has to understand the structure and the meaning of the database. Second, the user has to write a syntactic and semantic correct query statement that extracts the correct data from the database. In the next section we start with examples of different types of queries. Then we discuss some issues that are related to understanding the database.

2.2.2.5. Query Types

Reisner [Rei77] made a feature-by-feature analysis of two query languages. She tested languages on the following query types:

- *Simple mapping:* return the data values in columns which are associated with a known data value in another column. This is illustrated in the following query task: *Find the names of employees in department 50*.

```
SELECT NAME
FROM EMP
WHERE DEPTNO = 50
```

- *Selection (All Data in a row)*: to select all data values in a row, the names of the columns to be selected are omitted from the SEQUEL query, that is the FROM clause. This is illustrated in the following query task: *Give me the entire row of John Jones' employee record.*

```
SELECT EMP
WHERE NAME='JONES'
```

- *Projection (All data in a column)*: to achieve this, the WHERE clause has to be omitted. See the following query task: *List the names of all employees:*

```
SELECT NAME
FROM EMP
```

- *Assignment:* SEQUEL permits the user to assign query results to a new table as in the following query task: *Make up a new table with the name and the salary of each employee. Call the new table 'BUDGET'.*

```
BUDGET < - SELECT NAME, SAL
FROM EMP
```

- *Built-in functions:* the application of mathematical functions such as SUM, COUNT, MAX, MIN, and AVG. This is illustrated by the following query task: *Find the average salary of employees in department 50.*

```
SELECT AVG (SAL)
FROM EMP
WHERE DEPTNO=50
```

- *AND/OR* conjunction and disjunction may be used within a mapping, as in: *Find all employees who work for Mike Smith and who earn less than $20.000*

```
SELECT NAME
FROM EMP
WHERE MGR='SMITH'
AND SAL < 20000
```

- *Set Operations:* union, intersection, and set difference - between subqueries. For example, *Find the department numbers of departments located in Stockton which have employees who earn less than $5000*

```
SELECT DEPTNO
FROM DEPT
WHERE LOC='STOCKTON'
∩
SELECT DEPTNO
  FROM EMP
  WHERE SAL < 5000
```

- *Composition:* also called nesting of queries. The result of one mapping or query can be used as the input for another. This is illustrated in the following query task: *Find the names of all employees located in Stockton.*

```
SELECT NAME
FROM EMP
WHERE DEPTNO =
SELECT DEPTNO
FROM DEPT
WHERE LOC='STOCKTON'
```

- *GROUP BY:* this feature organizes the rows of a table into groups by matching the values in some of the columns, and then applies a built-in function to the rows in each group. See the following query task: *Find the departments with an average salary of over $15 000.*

```
SELECT DEPTNO
FROM EMP GROUP BY DEPTNO
WHERE AVG(SAL) > 150000
```

- *Correlation and computed variables* are features that were the predecessors of the JOIN operator. The correlation variable is a random chosen label (in the following example 'B1') that acts as a cursor to identify data values in successive rows in the outer (upper) block which must be correlated with the data values of the inner block. See the following query task: *Find the names of employees whose salary is bigger than their manager's salary:*

```
B1: SELECT NAME
FROM EMP
WHERE SAL>
 SELECT SAL
 FROM EMP
 WHERE NAME=B1.MGR
```

  The computed variables : values computed from one table can be displayed together with values obtained from another. This is illustrated in the following query task where the value of Q, the average salary obtained from the EMP table for a given department, is displayed together with the DEPTNAME information from the DEPT table. For example, *List the name of each department and the average salary of the employees in it.*

```
B1:SELECT DEPTNAME,Q
FROM DEPT
WHERE Q=
  SELECT AVG(SAL)
  FROM EMP
  WHERE DEPTNO=B1.DEPTNO
```

Reisner found that some of the language features were more difficult than others, namely:

1. *Correlation and computed variables*. This is clearly a very difficult feature, which caused a lot of errors. Errors indicated a basic misunderstanding of either the

syntax or the underlying concept. In SQL these functions are replaced by the JOIN feature.

2. *Group By.* Scores for Group By were low. Both groups however appeared to be able to use this feature, but only when they knew it was the one to use. This conclusion is based on Immediate Comprehension tests and on the first review after the feature was taught. The authors suspect that subjects attempt to translate from the English sentence to the Sequel query, looking for words in the sentence to serve as clues. The Group By is difficult because this feature is not mentioned explicitly in the English sentence.
3. *Composition.* Subjects had difficulties when the labels of the inner terms were different, with only the data types being the same. For example, a name of a person is in the first table denoted with the column NAME, in the second table with MGR. Reisner advises that database designers should use the same labels for the underlying data types. This is clearly a problem caused by the structure and naming of the data.
4. *Set operations*. Same phenomenon that is observed with composition.
5. *AND/OR:* there were a number of errors with these features, but the author thinks this is caused by an incorrect formulation of the specific question.

Schlager defined a model for the formulation of four categories of queries and validated this model with an experiment. Based on Reisner's findings, Schlager distinguished between the following categories of relational queries [Schl91]:

1. *Simple queries* request information from one table. In the simple conditional query the output form of the list items can be raw values or an arithmetic combination of two columns. One or more conditions (any combination of boolean operators) may be applied as long as the value of each condition is supplied in the query. An example would be, "Report the name and ID of everyone in department 20." Simple summary queries request a summary of all values in a column (minimum, maximum, sum, average, or tally). This category of queries is called simple in the query literature because most people can solve a reasonable query of this type with minimal training. They can become quite difficult to solve on the computer through the incorporation of many query language operators into a single query.
2. *Join queries* request list items of any form that reside in two or more different tables, under conditions with known values. These types of queries require that the database is structured in such a way that the two tables share a common column.
3. *Grouping queries* request a column summary (maximum, minimum, sum, average, or tally) for each value of a condition column.
4. *Unknown condition queries*, also called combination or composition queries, request raw list item values under a condition whose value is not specified in the query, and is therefore unknown. It requires the insertion of a summary or raw value from another table as the condition value.

The model that Schlager developed for the formulation process of these query categories is discussed in Section 2.2.2.7, “Query Formulation Process”.

2.2.2.6. Influence of the Database

The tasks discussed in the previous section cover the most important features of SQL. However, queries always operate on specific databases. There are a few indications in the literature that the database influences the efficiency and the effectiveness with which users write their queries. An indication can be found in the previous section. Reisner noticed that subjects had difficulties with formulating queries that included composition and set operators when the labels of the inner terms were different.

Lochovsky and Tsichritzis tested user performance of hierarchical, network, and relational databases and query languages [LT77]. Subjects were asked to produce queries for databases that were based on these data models. The user performance was measured by the amount of logical correct queries users could produce. Lochovsky and Tsichritzis remarked that there is always the danger that the application chosen might favor a particular data model. Therefore, they choose the following three very different applications and tried to balance the expected bias of each application separately.

1. Relational - airline schedules application
2. Network - medical records application
3. Hierarchical - election results of Canadian federal elections.

Based on the results of the experiment, they concluded that the relational system with the airline schedules application fared best in logically correct queries followed closely by the network system with the medical records application. The hierarchical system fared worst overall and for each application. They also state that the results also show that user performance of the data manipulation languages tested may be application dependent. In other words, the database structure and content may have influenced the user performance on the querying tasks.

Thomas and Gould observed in [TG75] that users often mismatched between terms used in the English question and terms used in the database. For example, for a question like 'Return all names of ...', several subjects typed in 'NAMES' while the appropriate column label was actually NAME. Greenblatt and Waxman found similar errors in [GW78] and, in addition, they noticed that it was a common error of subjects to put operators in non-numeric fields. A column named ITEM was often mistaken to hold the number of items, not the name of the item. Therefore, it is likely that database labels and possibly structure have an influence on the usability of query languages.

Brosey and Shneidermann [BS78] investigated the influence of the data model on user performance. This was tested with comprehension tasks, problem solving tasks and memorization tasks where no concrete query language was involved. Two databases

were used in the experiment: a relational database and a hierarchical database. The content and the application of the two databases were the same. The results of this experiment showed that the organization of the database makes a notable difference in performance of the subjects. In general, users performed better with the hierarchical database. This seems to be in contradiction with the results of Lochovsky and Tsichritzis that are mentioned above. However, there are two important differences between these studies. First, the studies use different databases and, as Brosey and Shneidermann mention themselves, other databases may lead to different results and conclusions. Second, the tasks in the experiments are different. In the study of Lochovsky and Tsichritzis, subjects formulated queries with a query languages, while in the study of Brosey and Shneidermann no formal query language was used. Here, subjects performed tasks that were related to question comprehension, problem solving and memorization. See also Section 2.2.2.4, "Query Tasks" for a definition of these type of tasks. We assume that these differences in databases and types of query tasks have lead to different results.

2.2.2.7. Query Formulation Process

In the previous section it was mentioned that query writing tasks consist of two components, namely understanding the database and formulating a correct syntactic construction. This can be considered as a very simple model of the human process of query formulation. A more complete model is needed to gain a better understanding of the query formulation process. In this section we summarize the proposed models from the studies mentioned earlier in this chapter.

Researchers who performed experiments to the ease of use of query languages, used a kind of qualification scheme for the output that the subjects produced. The qualifications in previous research vary from correct vs. incorrect to more complicated criteria for the correctness of the produced queries. We think that the error types that are distinguished do not only apply to the final result query, but also to intermediate query attempts that the user produces and improves until the final result. Thus, types of errors provide information about the query formulation process. We provide a small overview of the error criteria that are used in previous research.

The simplest system is to distinguish between correct and incorrect responses. However, this would mean that a basically good response with a small spelling error would be just as wrong as no response at all. A situation that is particularly troublesome in pen and paper tests where users do not receive direct feedback of their responses. Therefore, a number of researchers made a more detailed distinction: between logical correct and incorrect responses. This is distinction is made, for example, by Brosey and Shneidermann [BS78] and Lochovsky [LT77]. For logical correct responses small errors like spelling errors are ignored. A similar distinction is made by Greenblatt and Waxman [GW78], but in addition they made a category of no output and superfluous output. The latter is a correct query, but retrieves more data than is asked for. Reisner

et al. [RBC75] stressed some other aspects of erroneous responses. They devised the following scoring system, ranking from less serious error to serious error:

- *Completely correct:* the subject solved the query task correctly;
- *Minor data error:* the subject returned not entirely the correct result. For instance, only surnames when the sample database required the full name;
- *Minor language error:* the subject misspelt column names or omitted quotation marks;
- *Error of substance:* the subject produced valid query language expressions that would run, but produced the wrong answer;
- *Error of form:* the subject produced an invalid query language expression.

The advantage of this classification is that it is possible to distinguish between errors dependent on the query language and errors dependent on the database. Welty and Stemple used a grading method that was similar to Reisner's method:

- *Completely Correct;*
- *Minor language error:* the subject's solution was basically correct but had a small error that would be found by a reasonably good translator.
- *Minor operand error:* the solution has a minor error in the data specification, perhaps a misspelt column name.
- *Minor substance error:* the solution yields a result that is not quite correct but its incorrectness is due to the statement of the problem.
- *Correctable:* the solution is wrong but correctable by a good compiler.
- *Major substance error:* the query is syntactically correct but answers a different question than the one specified
- *Major language error:* a major error in the syntax (form) of the language has been made.
- *Incomplete:* incomplete query.
- *Unattempted:* no solution was attempted.

Welty and Stemple named the first four categories *essentially correct responses*. The other five were classified as incorrect.

Some models of the query writing process have been defined. In "Use of psychological experimentation as an aid to development of a query language" [Rei77] a preliminary model of query writing is suggested that is based on error analysis. This model assumes that a user translates the query task description to a query based on lexical items that are found in the description. Based on these items, a particular query template is chosen, for instance, a simple mapping query or a group by query. The appropriate database names are inserted into the query template based on the knowledge the users have about the database and the lexical items in the query description. Reisner assumes that some types of queries are more difficult to make for a user compared to other queries, because one query requires more operations to transform the lexical items

into a syntactic correct query. For instance, a group by expression has often no direct counterpart in the query task description.

Schlager [Schl91] performed a study to understand the cognitive processes that underlay database querying and thereby gained insights into the sources of difficulty faced by novice query language users. Schlager's model can be viewed as “simply a detailed and systematic description of what the user must know in order to perform the tasks”. Schlager formulated a GOMS model of the query formulation process. A GOMS model describes behavior in terms of four information processing components:

1. Goals: something that must be accomplished, or a task statement. Tasks are decomposed in goal hierarchies.
2. Operators: are actions performed in the execution of a task. Actions can be either mental or observable.
3. Methods: a sequence of operators used to accomplish a goal.
4. Selection rules: when more than one method can be used to reach a specific goal, a selection rule is used to choose one method.

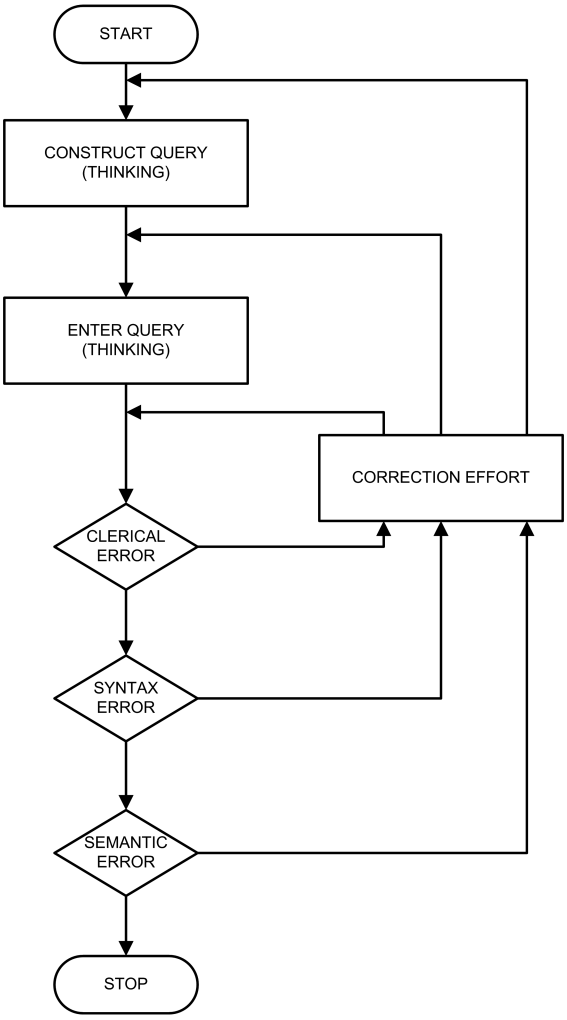
We provide a top level view of the query solving process according to Schlager. We only consider the goals and the selection rules used in the model, because the operators and methods are specific for the query language that is used.

1. Goal: solving query
    a. Goal: understanding the query
        i. Goal: establishing data table(s)
        ii. Goal: establishing conditions of search
    b. Goal: executing search of data.
    c. Based on the selection rule for the goal of executing search of data, one of the following goals is defined:
        i. Goal: executing simple search
        ii. Goal: executing join search
        iii. Goal: executing grouping search
        iv. Goal: executing unknown condition search

Schlager predicted performance differences between the four query types (simple search, join search, grouping search, and executing unknown condition search) with a different number of goals, operators, methods, and selection rules that are defined for each query. For instance, performance of users is better with simple search queries, because this type of query has less goals and methods compared to the join search.

Stohr et al. presented a simple query language interaction model to analyze the language's suitability for performing tasks [JV85]. This model is presented in the following figure:

Figure 2.1. Query Language Interaction Model



In this model, performance is determined by the time it takes to construct a query (thinking), writing a query (typing input), and correction of several types of errors. The thinking effort is what Reisner calls the selection of an appropriate template and what is presented by Schlager's process model. Input refers to the clerical effort: typing in the query. Three types of errors are mentioned:

1. Clerical errors, e.g. typos;
2. Syntactic errors, not following the correct syntax of the language;
3. Semantic errors, formulation of a syntactic correct query that does not solve the correct problem.

Based on the models mentioned in this section, we present our hypothetical model in the next chapter.

2.3. Conclusions

Until now, the focus on XML query language development has been on the functionality of languages and implementation issues. Three potential important query language for XML are XSLT, XQuery, and SQL/XML. These language differ in several aspects. For instance, XQuery is claimed to be more suitable for optimization for use with an XML database. SQL/XML is more suitable in combination with a relational database than XSLT. Despite these differences in machine effectiveness, the languages share the same basic functionality.

There is some discussion about usability aspects of XSLT and XQuery. XSLT is considered to be difficult to learn due to the functional programming character and the verbose XML syntax. On the other hand according to XSLT adapts, XQuery adds no functionality to XSLT and is therefore considered to be more or less redundant. Also, they claim that XQuery is mainly suitable for data-oriented XML documents, but XQuery adapts disagree with this claim. SQL/XML is still very early in its development and not widely used. Therefore, SQL/XML is not discussed often in the XML community yet. However, SQL/XML is more verbose than XQuery due to the extensive use of conversion functions but still less verbose compared to XSLT. We noticed that the naming conventions of the SQL/XML extension functions are not very consistent. SQL/XML is an extension of SQL and therefore we expect that users of SQL will have little difficulties with learning SQL/XML. The similarities and differences between XSLT, XQuery, and SQL/XML are summarized in Table 2.1, “Feature Overview”.

The focus of this study is on the novel area of usability of XML query languages. Some research has been carried out to the ease of use of relational query languages, and the following factors were identified as having influence on the performance with a query language:

1. The query language: performance differences were found for the same set of tasks with different languages. This means that the use of different query languages can lead to measurable performance differences. Little is known about specific language features that influence query performance. In general, only very general notions like textual vs. graphical languages are used to characterize a language and explain performance differences. Welty and Stemple use the notion of procedurality that is based on the number of expressions, variables bindings, and their order in a query expression. This procedurality metric is difficult to apply to other languages, because the exact definitions of these terms cannot be translated directly to, for instance, XSLT and XQuery.

2. User experience: it was found that user experience had an influence on the query process. User experience is defined in terms of programming experience and/or in terms of database experience. In general, more experienced users performed better than less experienced users. Some exceptions to this rule were also found and this was explained by assuming that previous experience with a completely different language conflicted with learning a new language.
3. The query type also has an influence on the query process. In the experiment of Welty and Stemple, performance differences with relational query languages were only apparent with the difficult queries and not with simple queries. In addition, it is noticed by several researchers that performance on simple queries was higher compared to more complex queries. Simple queries include queries that select data based on one or more criteria, complex queries are joins, grouping, and combinations of different query types.
4. The influence of the content and structure of the database is not investigated thoroughly yet. Most experiments used rather simple databases that were easy to understand for the subjects. However, some indications were found that the performance with a particular query language is influenced by the data structure.
5. Based on error analysis of the queries, some preliminary models of the query process have been proposed. The GOMS model of Mark Schlager for four types of queries is probably the most complete.

In the next chapter we propose a more general model of the query process that we think applies to all types of queries. We will use this model in a thinking aloud experiment to gain a better understanding of the behavior of users querying with an XML query language.

Chapter 3. Understanding the Query Process

3.1. Introduction

In this chapter we describe our assumed model of the query solving process for XML query languages. The sources of the model are discussed, as well as the definition of terms used in the model. Second, the factors that are discussed in Chapter 2 'Background', are related to XML Query languages and the model of the query solving process. Important questions are: why do these factors probably also apply on the ease of use of XML query languages? What is the relation with the query solving process? Based on this discussion, the central research questions of this study are formulated. We conclude with an overview of the required experiments that are needed to answer the research questions.

3.2. Model of the Query Solving Process

3.2.1. Introduction

In this section we describe an assumed model of the user's query solving process that will be explored in the first experiment[1]. The model is based on four sources. First, the query language interaction model of Stohr et al. 1982 [JV85]. Second, the GOMS model of Schlager [Schl91]. Third, the error scoring methods that are discussed in the previous chapter. Fourth, our own personal experience and a pilot test with one subject.

A common idea of earlier models is that the query solving process consists of two main activities. This is also suggested in the error classifications when a difference is made between syntactic incorrect queries and semantic incorrect queries. The first activity is about understanding the problem that needs to be solved and the second activity is about formulating this problem into a syntactically correct query expression. The terms we use are the same as Schlager uses for these activities, namely *Construct Deep Structure* and *Construct Surface Structure* respectively. More specific assumptions about these activities are the distinction between *Understand the Conditions of Search* and *Understand the Data Table(s)*. The *conditions of search* includes a classification of the problem that needs to be solved. It also includes the

[1] This model is earlier described in [Gra04]

decomposition of an overall problem in smaller subproblems and the selection of a subproblem that is to be solved first. *Understand the data tables* means that the user tries to understand how the items in the problem definition are related to the database structure and names. The deep structure can be understood as a detailed problem definition and the surface structure as the matching query language expression.

In the model of Stohr three types of correction efforts are included: clerical errors, syntactic errors, and semantic errors. In our model, we extend the correction effort with more types of user errors based on the error types distinguished by Reisner and Welty and Stemple and on some additional error types that we found during a pilot test with the model. For instance, interaction with a query processor is not explicitly included in the models and neither were types of errors that were related to procedural mistakes like forgetting to submit the query to the processor. We also included activities that are related to the understanding of the query descriptions and to evaluating the information provided by the query processor. The main difference with the models of Reisner and Schlager, is that our model describes the iterative process that underlies all query formulation efforts while Reisner and Schlager modeled the user's effort on specific types of queries like simple selection queries, join queries and grouping queries.

The model we present here is intended as a universal and detailed description of how we assume users solve query tasks. In the previous chapter, we discussed earlier research where user performance differences were found between different relational query languages. In the XML community similar user performance differences are assumed between XML query languages. When the use of different query languages results in performance differences, the way the queries are solved with these query languages must be different as well. The model of the query solving process describes how queries are solved. Our main and most important assumption is that the performance differences with query languages can be understood in terms of the query solving process.

The process of solving a query task consists of several activities and subactivities. We assume that the number, and possibly the order, of the user's activities during the query solving process is influenced by the following factors:

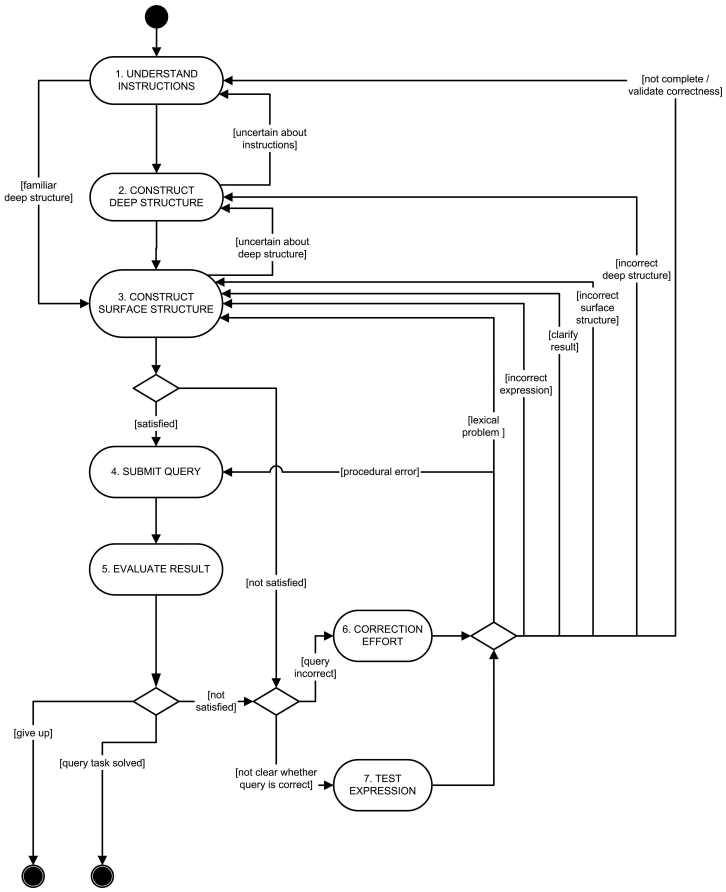
- Syntax of the query language, in particular the procedurality, verboseness, and the consistency of the syntax;
- Query type, or the problem description and problem type;
- Data structure;
- User experience;
- Query processor;
- Query interface;
- Quality and volume of reference materials.

In the following sections we provide diagrams and a specification of each activity of the assumed query solving process. In Section 3.3, “Influences on the Query Solving Process” we discuss the exact assumed relations between these factors and the query solving process. The model of the process enables us to interpret the behavior of querying users in our experiments and, possibly, relate performance differences caused by query languages to the number and order of activities in the model.

3.2.2. Top Level Activities

We start with an overview of the top level activities and then continue with a more detailed description of the subactivities in the model. We use a UML activity diagram to describe the model. The flattened ellipses represent activities of the user. This can be an action, or a thinking or reasoning step of the user. Activities can consist of subactivities which will be described later on in subdiagrams. The start of the query solving process is represented with a black dot, the end of the query solving process is represented with a black circled dot. The diamond boxes represent a decision with two or more possible transitions, or a merge point where multiple branches come together.

Figure 3.1. Query Solving Process: Top Level



The query solving process consists of the following top level activities:

1. *Understand instructions.* To solve the query correctly, the user must understand the instructions that describe the query task that needs to be solved. Based on the instructions, the user either has a global, or an exact idea of the query problem. When the user has a global idea of the query problem, some aspects need further specification and this is done during the activity of constructing the deep structure. When the user has an exact idea of the problem, this means that the deep structure of the query task is familiar to the user and he continues to construct the surface structure of the query task.
2. *Construct Deep Structure.* The deep structure of a query task is a complete and thorough understanding of which data must be retrieved from the XML collection and how the result data must be structured. The deep structure of a query task is independent of the query language that is used. When the user fully understands

which data must be retrieved and how the data needs to be returned, he starts to construct the surface structure of the query task. During the construction of the deep structure, it is possible that the user is uncertain about the instructions and returns to the activity of understanding instructions. For instance, to check for the name of an element.

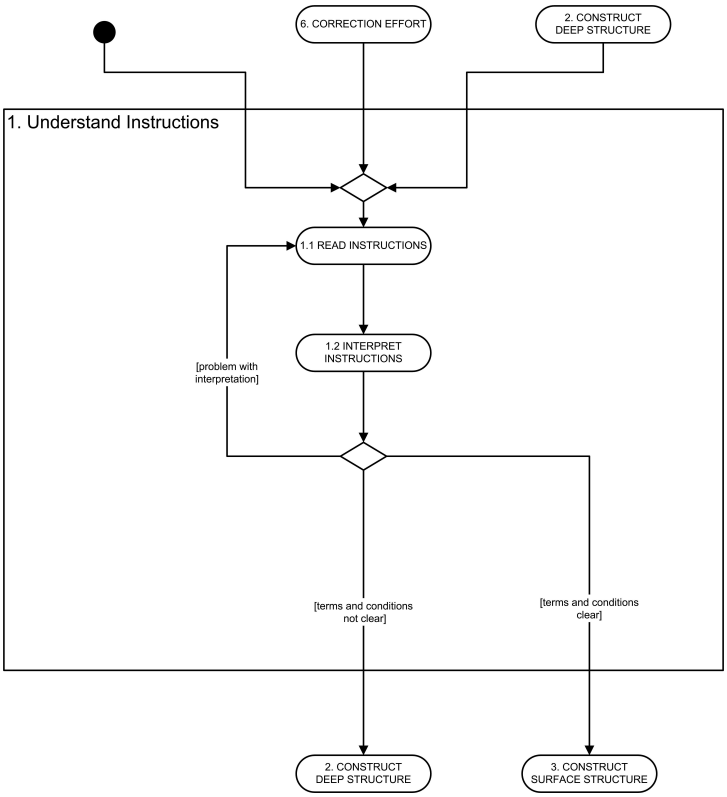
3. *Construct Surface Structure.* The surface structure of a query task is a query language expression that is a translation of the deep structure of the query task. Once the user is satisfied with the query expression, he submits the query to the processor. When the user is not satisfied, he attempts a correction effort or tests the expression. The user returns to construct the deep structure of the query when he is unsure about the problem that needs to be translated. For instance, when a subproblem is no longer clear to the user, or when the user wants to translate another subproblem.
4. *Submit query.* The user submits the query language expression to the query processor.
5. *Evaluate Result.* The user reads and interprets the information he receives from the query processor. This information consists of possible error messages and/or the resulting XML document of the query expression. When the user thinks he solved the query correctly, he stops and the query solving process ends. When the user is not satisfied with the result, he either continues to test the expression, makes a correction effort, or gives up to formulate a solution.
6. *Correction effort.* The user tries to understand what type of error is made with the query expression. Based on this understanding he returns to one of the previous activities.
7. *Test expression.* This activity is similar to a correction effort, only in this situation the user thinks the query expression is correct.

In the following section we discuss the individual subactivities in the model in more detail.

3.2.3. Understand Instructions

The activity of understanding the instructions consists of two subactivities: reading the instructions and interpreting the instructions. The activity is represented in the diagram below:

Figure 3.2. Understand Instructions

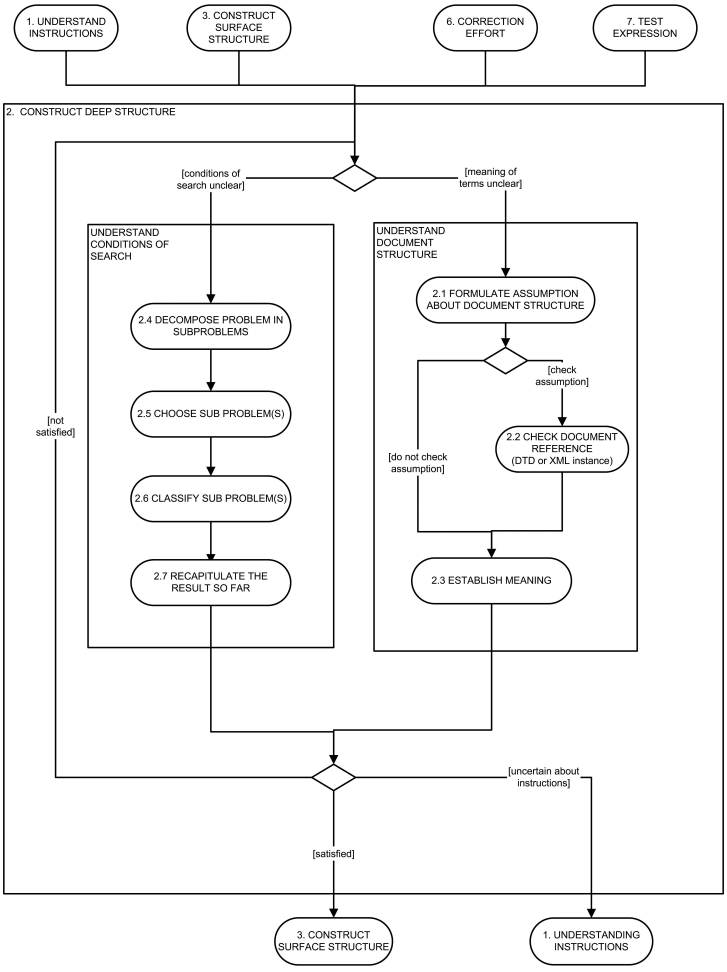


When a user has problems with the interpretation of the instructions, for instance when the query task description is complex or unclear, he can read the instructions again until he has a global understanding of the query task. Once the user understands the instructions, he continues to either the construct deep structure activity or the construct surface structure activity based on the extent to which the user understands the problem. For instance, when all terms and all conditions in the instructions are clear to the user, he continues to construct the surface structure. When the terms and conditions are not fully clear, he continues to construct the deep structure.

3.2.4. Construct Deep Structure

The deep structure of a query task is a complete and thorough understanding of which data must be retrieved from the XML collection and how the result data must be structured. The construction of the deep structure can be divided into two subactivities: understand the search conditions and understand the document structure. Also see the following diagram:

Figure 3.3. Construct Deep Structure



We will discuss both activities below.

3.2.4.1. Understand Conditions of Search

The *conditions of search* is the underlying problem of the query task. The following user actions are part of the activity of understanding the conditions of search. These activities are based on general problem solving strategies that are mentioned in [LN77], chapter 14.

- [2.4] *Decompose problem in subproblems:* the user divides the overall problem in smaller subproblems. For instance, when a user needs to select all male persons

older than 20 years he can decompose this problem in "select all persons", "get all males", and "get everybody in this group older than 20". Of course, for most problems several decompositions are possible.
- [2.5] *Choose a subproblem:* the user chooses a particular subproblem that he wants to solve first. In the example above, he can choose to first solve the query of selecting all persons before continuing with the other subtasks.
- [2.6] *Classify subproblems:* the user recognizes a problem or subproblem as being a similar other problem. For instance, he recognizes a problem as an instance of a problem described in a tutorial, as an earlier solved problem, or as a typical query type like a join.
- [2.7] *Recapitulate the results so far:* the user recalls which subproblems there are, and which subproblems have not been solved yet.

In the model we represent the *construct deep structure* as an iterative activity. Note that the order of subactivities during the activity of understanding the conditions of search are not as strict as shown in the diagram. Once the user understands the conditions of search, possibly of a subproblem, he either continues to the activity of *understand the document structure*, or to the *construct surface structure* activity.

3.2.4.2. Understand Document Structure

In addition to the conditions of search, the user must know the parts of the data structure that are relevant to these conditions. For instance, if the user needs to find all male persons older than 20 in the document, he needs to know how persons, sex, and age are represented in the document schema. The user can perform the following actions to gain understanding of the document structure:

- [2.1] *Formulate assumption about document structure:* the user makes an assumption about how the data he needs to find is represented in the XML document. This assumption can be related to both element and attribute names used in the document, and to the structure of the document. An example of the former is when the user wonders which element name is used for a person. An example of the latter is when the user wonders whether the age of a person is represented as an element or as an attribute.
- [2.2] *Check document reference:* the user reads a reference about the document structure. For instance, the DTD or XML Schema, the XML document instance itself, or a description of the document schema.
- [2.3] *Establish meaning:* the user makes a conclusion about the XML document structure. For instance, that a person's age is represented with the attribute year.

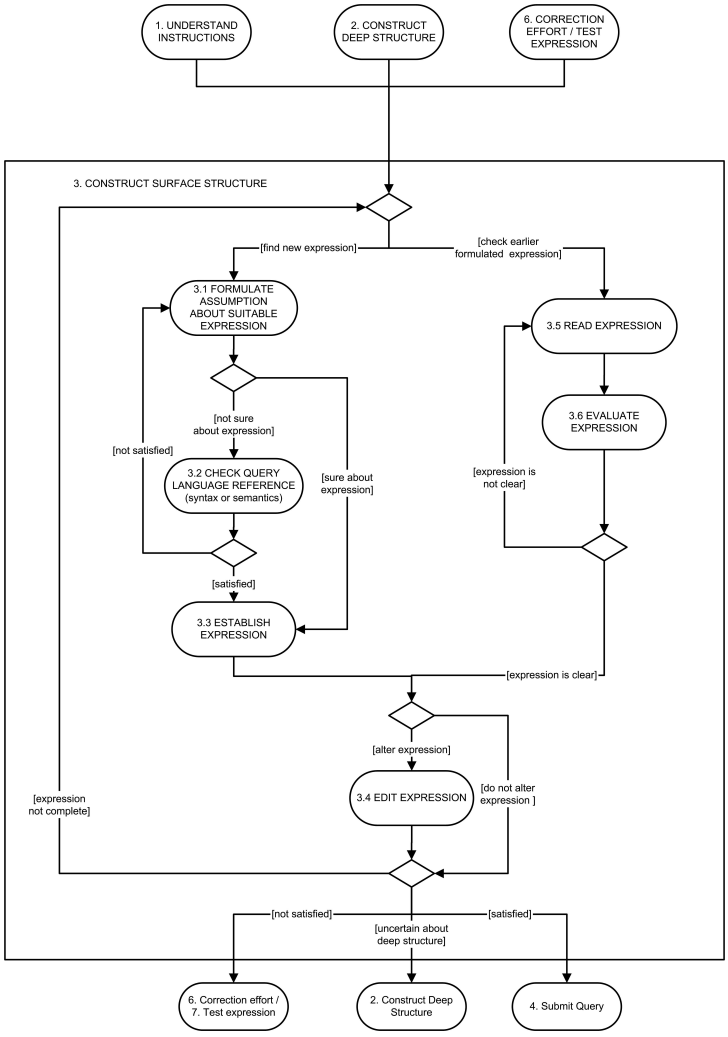
During the activity of constructing the deep structure, it is possible that the user becomes uncertain about the instructions and returns to the activity of understanding the instructions. For instance, when he forgot whether he should return the age of male or female persons in the document. When the user understands the part of the document

structure that relates to the problem he wants to solve, he continues with constructing the surface structure of the query task.

3.2.5. Construct Surface Structure

The surface structure of a query is the translation of the deep structure to an query language expression. During this activity, the user translates a subproblem that is constructed in the deep structure phase into a suitable query language expression that can be submitted to the query processor. See the diagram below:

Figure 3.4. Construct Surface Structure



The following subactivities are a part of the construction of the surface structure:

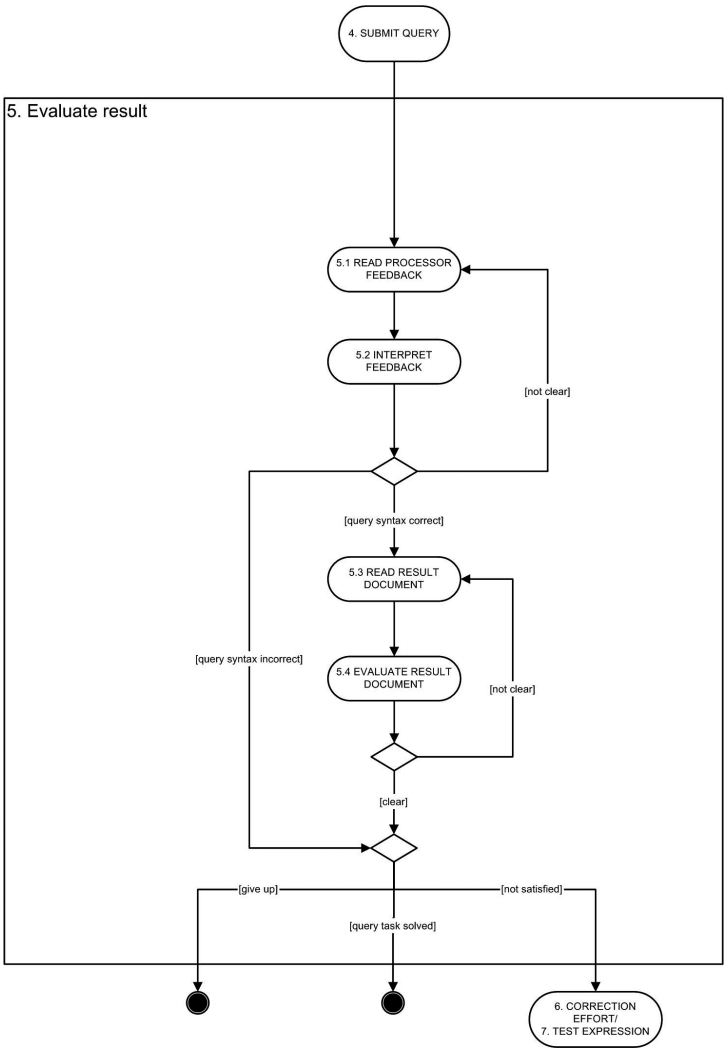
- [3.1] *Formulate assumption about suitable expression:* the user makes an assumption about a suitable expression, or wonders which syntax construct he can use to solve the problem.
- [3.2] *Check query language reference:* the user checks a reference guide or a tutorial about the query language to learn more about a specific syntactic construct.
- [3.3] *Establish expression:* the user determines which syntax construct he will be using to solve a particular subquery. This can be done after checking a manual about the language, or based on his experience with the language.
- [3.4] *Edit expression:* the user types a query expression in the interface. Cut and paste from a tutorial or a previous query is also considered as editing.
- [3.5] *Read expression:* the user reads a previously formulated expression. This is an expression that the user wrote earlier during the query solving process. Therefore, this activity can only occur when the user already formulated an expression.
- [3.6] *Evaluate expression:* the user tries to understand what the result of the expression will be.

If the user is satisfied with the query expression, he submits the query to the processor. If the user is not satisfied he either tests the expression or makes a correction effort. When the user has doubts about the deep structure of the query, he returns to the activity of constructing the deep structure.

3.2.6. Evaluate Result

After the user has submitted the query to the processor, the user reads and interprets the error messages and/or the result document. This is represented in the diagram below:

Figure 3.5. Evaluate Result



Evaluate result consists of the following subactivities:

- [5.1] *Read processor feedback:* the user reads the messages of the query language processor. The query processor can return several types of messages, for instance: the query is executed or the query consists of a syntax error in line 6.
- [5.2] *Interpret feedback:* the user tries to understand the feedback of the processor.
- [5.3] *Read result document:* a result document is only returned when the query was syntactically correct. If this is the case, the user reads the result document.

- [5.4] *Evaluate result document:* the user relates the result document to the deep structure problem that he intended to solve. Based on this evaluation he decides whether the query task is solved correctly or incorrectly and how to proceed.

When the user has understood both the feedback of the query processor and the possible result document, he decides if the result is correct or if the query needs to be corrected or tested. At this point, it is also possible that the user gives up and quits the query solving process.

3.2.7. Correction Effort / Test Expression

The user needs to make a correction effort when he noticed an error in a query expression or when he wants to improve the query. To improve the query, the user must have a certain level of understanding of what is wrong with the current expression. The following types of errors or problems are included in the model. See Figure 3.1, "Query Solving Process: Top Level" for a graphical representation.

- *Procedural error:* the user suspects that the query is not correctly submitted to the query processor due to a technical problem. For instance, because a button is not pressed correctly. In this case, the user will probably try to submit the query again or try to solve the technical problem. Note that strictly speaking this is not related to a query language, but to the interface that is used.
- *Lexical problem:* the user notes that he made a typo. For instance, he forgot a comma or a bracket. The user returns to the surface structure to edit the query. When a query with this type of error is submitted to the processor, it will result in a syntax error.
- *Incorrect expression:* the user notes that he used an incorrect expression. For instance, he has placed the ORDER BY clause before the WHERE clause in a query. When a query with this type of error is submitted to the processor, it will result in a syntax error.
- *Clarify result:* the user is not satisfied with the layout of the result and tries to improve this by editing the query.
- *Incorrect surface structure:* the user notices that the result document that is returned by the processor, is not the result that he expected. When a query with this type of error is submitted to the processor, no error messages are returned. The query is syntactically correct, but does not provide the expected result. The user returns to construct the surface structure to correct the query. Note that in this situation the user has a clear understanding of the deep structure, so the problem must be located in the surface structure of the query.
- *Incorrect deep structure:* the result of the query is the expected result of the user. However, the user realizes that this result is not the correct result of the query task. For instance, the user formulated a query that returns a list of year attributes, but the user suddenly realizes that this was not according to the original query task.

He returns to construct the deep structure activity, for instance to recall that he needed a list of persons.
- *Query incomplete:* the user successfully created a query that solved a particular subproblem. He returns to understand the instructions to proceed with the next problem.
- *Validate correctness:* the user wonders whether he solved the query correctly and returns to the activity of understanding the instructions to compare the obtained result with the query task description.

Now that we described our assumed model of the query solving process, we will continue to discuss the factors that may influence this process in the following section. In addition, we formulate expectations how these factors influence the query process. The assumptions will be expressed in the relative numbers of (sub-)activities and the direct and indirect transitions between activities. Indirect transitions include other activities and we refer to them as 'paths'.

3.3. Influences on the Query Solving Process

In this section we discuss the relevance of the findings from the previous chapter to the usability of XML query languages. We formulate assumptions about query language properties and their influence on the query solving process. We will also discuss other, non-query language related, relevant factors. We express our assumptions about query language properties in the number of activities, and transitions and paths between activities in the model and in the number of specific errors.

3.3.1. Syntax Properties

3.3.1.1. Procedurality

In the previous chapter we have seen that human performance differences were observed with different relational query languages. Performance is usually measured as the correctness of the result queries and the time that subjects needed to reach a solution. We think that querying a relational database is comparable with querying an XML document because the types of problems are quite similar: extracting data from a particular data source with a specialized formal language. Therefore, differences in performance and models that describe the query solving process for relational databases will probably also apply to XML query languages. Unfortunately, no satisfactory explanations were given in previous research to account for the relation between query language constructions and performance differences. In most experiments, one specific property of a language was equated with the language itself. For instance, SQL is considered as a textual language and QBE as a graphical language and the difference

between language performance was explained by the difference between a textual and a graphical language. In our research, we could use a similar distinction between XSLT and XQuery by labeling them as having an XML syntax and a SQL-like syntax respectively. Although not incorrect, we think that such a distinction is too general to be meaningful.

Instead, we want to explain performance differences between XML query languages by their distinguishing properties. For instance, Welty and Stemple [WS81] distinguished between procedural and non-procedural relational query languages. Procedurality is the degree to which a programming language expression describes *how* or the *method* to obtain a desired result, instead of describing the desired result itself. This latter is called non-procedural. Welty and Stemple argued that a procedural problem statement fits better with the natural problem solving strategies of humans compared to non-procedural problem statements. The difference in procedurality between two languages was assessed with a procedurality metric. According to Welty and Stemple, the intuitive idea of procedurality of a language expression is determined by the number of variable bindings, the number of operations, and the degree to which the bindings and operators are ordered by the language semantics. Their procedurality metric quantified this intuitive idea of procedurality of a language. However, this metric is rather specific for the query languages they used and cannot be applied to XML query languages. On the other hand, it seems to be intuitively sound to consider both XQuery and SQL/XML as being more procedural than XSLT. The meaning of an XSLT program does not change when different template rules are placed in a different order. In contrast, it is not possible to change the order of keywords of XQuery and SQL/XML expressions.

An argument that is similar to Welty and Stemple's hypothesis is made in the XML community, in particular by XQuery supporters. XSLT is considered as more difficult to learn compared to XQuery. One of the reasons for this is the functional, declarative programming paradigm of XSLT. This paradigm is instantiated in XSLT with template rules. Based on these arguments, we assume that one of the relevant properties of the XML query languages that may influence performance is the degree of procedurality, where a more procedural language fits better with the natural manner of human problem solving. If this assumption is correct, there must be a difference how queries are solved with a procedural query language and a less procedural query language. Welty and Stemple's query solving strategies correspond to the conditions of search activities in our model, which are subactivities of the understand deep structure activity. The difference between procedural and non-procedural problem statements corresponds to expressions that are formulated during the construct surface structure activity. Welty and Stemple's hypothesis can be interpreted as follows: the deep structure is easier to express in the surface structure with a more procedural language and more difficult to express with a less procedural language. For instance, a user decomposes a query problem in three subproblems during the construction of the deep structure. With a procedural language it may be easy to express and combine these three subproblems in corresponding subexpressions. This may be more difficult with a less procedural

language, for instance because the three subproblems must be expressed in one expression.

Therefore, users of a less procedural language will encounter more difficulties during the construction of the surface structure and they will have more transitions and paths between constructing the deep structure and the surface structure of a query. The paths include other subactivities, for instance when a user formulated a query expression, submits the query to the processor, evaluates the result, and returns to the activity of constructing the deep structure. As a result of this, we expect that for more procedural languages the number of construct deep structure (in particular the *understanding conditions of search* subactivities) and surface structure subactivities needed to solve a problem are smaller than for less procedural languages. Also, we expect two types of correction efforts to occur more often with less procedural query languages. First, correction efforts of errors related to the deep structure. Second, correction efforts of errors that are related to the translation from deep structure to surface structure, or the surface structure error. This type of error means that the query language expression is not a correct translation of the deep structure of the query.

3.3.1.2. Verboseness

Another argument from the XML community is that XSLT is more difficult to use compared to XQuery due to the verboseness that is inherent in XSLT's XML syntax. Examples of verboseness in XSLT are the namespace declarations, the opening and closing tags, the pointy brackets, and so on. The assumption that a more verbose language has a negative effect on user performance is also mentioned in another context, namely about programming languages. Gupta states that, in particular for new users, the less lengthy Python programs are easier to write and understand compared to more verbose Java programs [Gu04]. However, up to now we have found no empirical studies that support the assumption that a more verbose query or programming language results in lower user performance. Languages can differ in verboseness for several reasons. For instance, Java is a strongly typed language and, according to Gupta, this imposes a lot of restrictions to the user that result in lengthy Java code. For XSLT, the reason for the verboseness is the use of XML syntax. Regardless of the reasons, verboseness is generally considered as the length of language expressions. Other measurements we might use for verboseness is the number of query language operators. A similar measurement is used in psychological studies of text comprehension, namely the number of words in a text. However, we consider verboseness simply as the length of, or the number of characters in a query expression, because in our opinion this measurement fits better to the informal notions of verboseness than the number of query language operators.

The rationale of the argument that verboseness has a negative effect on user performance, is that verbose syntax makes it more difficult to formulate the surface structure of a query. If verboseness is a factor that influences the query solving process,

there will be differences in how queries are solved with a more verbose language and a less verbose language. Verboseness is related purely to the syntactic form of a query language. Therefore, we expect that the differences caused by verboseness will become apparent in subactivities that are related to the construction of the surface structure. Also, we expect that errors related to the syntactic form of the query expression will occur more often while using a more verbose language. These are the following errors: lexical problems, incorrect expressions, and clarify result. In addition, these errors and the difficulties users may encounter during the construction of the surface structure will result in more loops of surface structure, submit, evaluate, correction, surface structure.

3.3.1.3. Consistency

Nielsen states that "consistency is one of the most important usability characteristics" for user interfaces [Nie93]. Consistency is informally described as that the same command or the same action will always have the same effect. We noticed that SQL/XML is less consistent compared to both XQuery and XSLT due to the use of two different data models and the potential confusing naming conventions of several functions. The consistency of a language is similar to a factor like verboseness with respect to the influence of the query solving process. Both are related to the formulation of the surface structure of the query. Therefore, inconsistencies in a language will have a similar effect as verboseness of language. This means that the effect of inconsistencies will result in a larger number of activities of constructing the surface structure, surface structure related correction efforts, and loops from surface structure to surface structure.

3.3.2. Complexity of a Query

In Chapter 2, *Background* we discussed that large differences have been found on the performance on different types of queries like simple selection queries, join queries, and grouping queries. Schlager [Schl91] compared the performance on four types of queries, where the least complex queries were named simple queries and more complex queries join, grouping, and unknown condition queries. Schlager showed that performance differences on different query tasks could be predicted with GOMS models of these tasks. See Section 2.2.2.7, "Query Formulation Process" for a brief description of these models. Performance on a specific query could be predicted with the number of steps in the model of the task. For instance, it was noted that performance was the best with the simple queries because the simple queries could be modeled with a few steps. Schlager modeled more complex queries as a combination of multiple simple queries and therefore the models of more complex queries consisted of more steps than the less complex queries. Our model of the query process is independent of the query type that is used. The difference between less complex and more complex queries, is noticeable by the number of iterations over subproblems. Schlager did not compare multiple query languages. However, in other previous studies, it was found

that performance differences between query languages became more apparent with more complex queries. For instance, Welty and Stemple [WS81] found only performance differences between two languages on difficult queries, and not on simple queries.

We assume that the query type influences the whole query solving process. More complex queries will result in an overall larger number of subactivities of the query solving process. For instance, a query task description for a more complex type of query will be more difficult to understand compared to a simple query. Therefore, the activity of understanding the instructions will occur more often with complex queries. Also, the deep structure of a complex query task is more difficult than for simple queries. Complex queries can be expressed as a combination of multiple simple queries where the result of one simple subquery is an argument for another simple subquery. Therefore, the subactivities within constructing a deep structure will occur more often. Finally, more complex queries need more syntax constructions compared to simple queries. Thus, there also will be a larger number of construct surface structure subactivities with complex queries and for this reason also more correction efforts will be required. Because of these reasons, the performance differences between languages will probably become more apparent with more complex queries. The overall query solving process will include more subactivities for complex queries and the differences that are caused by language properties will therefore become stronger than with less complex queries.

3.3.3. Data Structure

It is generally assumed that XML document structure is an important factor when querying XML data. Data structure is a more general term for XML document structure that can also be applied to relational databases. In previous research, only a few indications have been found that the content and structure of the database influence the performance with a query language. These indications were mainly related to discrepancies between column names in the database and terms used in the query task descriptions. Although this is an important practical issue, it is not directly related to the influence of different XML structures on the query solving process and it is not related to query language properties.

In the XML community, XSLT advocates argue that XSLT is more suitable for document-oriented XML applications compared to XQuery. This type of application is characterized by XML structures that have, in contrast to data-oriented XML applications, a more irregular structure, more mixed content, and the content is in general textual information for human consumption. XSLT's template rules are supposed to make processing of document-oriented XML data easier. Based on these indications and assumptions, we formulate the following possible influence of data structure on the query solving process. If the arguments of the XSLT advocates are correct, XSLT users will need less transitions and paths between constructing the deep

structure and constructing the surface structure for document-oriented XML applications. These paths include, for instance submitting the query to the processor, evaluating the result, and correction efforts. Therefore, we assume that the number of construct deep structure activities needed to solve a problem, most notably understand conditions of search subactivities, are smaller than for XQuery. Also, we expect that two error types will occur more often with XQuery. First, errors related to the deep structure. Second, errors related to the translation from deep structure to surface structure, or the surface structure error. Note that we earlier stated that the number of transitions between deep structure to surface structure would possibly be lower for procedural languages. However, if we include data structure and focus on a specific difference between XQuery and XSLT we expect the opposite.

3.3.4. User Experience

Several experiments indicate that users with programming experience perform better with relational query languages than users with no, or with less, programming experience. In the XML community it is claimed that XSLT is difficult to learn by users who have experience with non-functional programming languages, but there is no empirical data that supports this idea. We assume that for the following reasons experienced users will, in general, perform better than less experienced users. Experienced users have solved similar problems more often compared to less experienced users. Therefore, experienced users will understand the instructions and the deep structure of a problem faster than less experienced users. Also, the formulation of a surface structure and the debugging of queries will be easier for more experienced users. In short, we assume that the more experienced a user is, the less iterations over the query solving process will be needed to formulate a correct query. There may be two situations where more experienced users perform worse than less experienced users. First, as stated above, users may be experienced with one type of language and encounter difficulties with other types of languages. For instance, users who are highly experienced with a procedural language may encounter difficulties when using the more declarative language XSLT. For instance, they will encounter more difficulties during the construction of the surface structure and they will have more transitions and paths between constructing the deep structure and the surface structure of a query. In other words, in this situation we expect a larger number of the following activities: construct deep structure, and in particular the understand conditions of search activities, construct surface structure, submit, evaluate result, and correction efforts of surface and deep structure errors. Also the number of transitions and paths between the construct deep structure and construct surface structure will be larger. Second, it is possible that experienced users will perform less than less experienced users with simple queries. This was noted by Catarci and Santucci [CS95]. They assumed that the reason for the lower performance was that experienced user underestimated the simple query and formulated their query too hasty. If this is correct, we expect that the number of construct deep structure activities will be lower for experienced users working on simple queries.

3.3.5. Other Influences

We expect that there are other influences on the query solving process besides the factors that are discussed in the previous sections. For instance, we assume that the query processor influences the query solving process. A fast query processor may stimulate the user to submit partially solved queries more often compared to a slow query processor. Also, the quality of the query processor's error messages is important. When the processor delivers very specific and clear error messages, the user's correction efforts will be easier compared to a query processor with none, or less clear error messages. Another factor that may influence the query solving process is the interface of the query processor. It is likely, for instance, that a query editor with syntax highlighting leads to higher performance compared to an editor without syntax highlighting. Another factor that can influence the query solving process, is the quality of the available reference materials. We assume that users perform better with high quality reference material.

In this section we discussed our assumptions about how several factors may influence the query solving process. These factors came from previous research on the ease of use of relational query language and common arguments about XSLT and XQuery from the XML community. In the following section we discuss the research questions of this study and we provide an overview how we intend to answer these questions.

3.4. Research Questions

The main purpose of this study is to understand differences in usability between several XML query languages. We provided an overview in Chapter 2, *Background* of current XML query languages, an introduction to aspects of usability engineering, and an overview of previous relevant research. In this chapter, we formulated an hypothetical model of the query solving process that applies to querying information sources in general, and to XML documents in particular. Also, we discussed how several factors may influence this query solving process. Our primary research topic is the influence of query language aspects on the user performance with that language, and the satisfaction of that language. Secondary, we are interested in other factors that lead to performance differences with XML query languages. For instance, user experience and document structure. Factors that are likely to influence user performance, but are outside the scope of this study are query processor, query interface, and query language reference materials.

We now formulate the research questions that are based on the initial goal of this study and the assumptions mentioned in the previous section. First, the global research questions of this study are as follows:

1. What are the differences in user performance between XQuery, SQL/XML and XSLT on a set of representative query tasks?
2. What are the reasons for performance differences?
3. Which language results in the highest user's satisfaction?

We expect that the following factors result in performance differences:

1. User performance is better with a more procedural query language like XQuery and SQL/XML, than with a less procedural query language like XSLT. More procedural languages will result in a lower number of the following activities, transitions, and paths:
   - Construct deep structure, and in particular the Understanding conditions of search.
   - Construct surface structure
   - Correction effort, Deep Structure
   - Correction effort, Surface Structure
   - Deep Structure > Surface Structure
   - Surface Structure > Deep Structure
   - Construct Surface Structure > Submit > Evaluate Result > Correction Effort or Test Expression > Construct Deep Structure

   An important characteristic of the non-procedural nature of XSLT are the template rules. These template rules may improve user performance on document-oriented XML applications. Therefore, for document-oriented XML applications we expect that these number of activities, transitions, and paths will be lower with XSLT.
2. User performance is better with a less verbose language like XQuery and SQL/XML, than with a more verbose language like XSLT. A more verbose language will result in a higher number of the following activities, transitions, and paths:
   - Construct surface structure
   - Correction effort, Lexical problem
   - Correction effort, incorrect expressions
   - Correction effort, clarify result
   - Construct Surface Structure > Submit > Evaluate Result > Correction Effort or Test Expression > Construct Surface Structure
3. User performance is better with consistent languages than with inconsistent languages. We noticed some inconsistencies in SQL/XML and these may decrease user performance. Consistency of a language has the same effect on the number of activities, transitions, and paths as verboseness.
4. Differences between query languages become more clear with more complex queries than with less complex queries. More complex queries will result in an overall higher number of activities, transitions, and paths during the query solving process.
5. Users who are more familiar with a particular type of query language will perform better with a similar language. These users will have an overall lower number of activities, transitions, and paths during the query solving process.

3.5. Overview of the Experiments

Little is known of the factors that influence XML query language usability. In literature, we found a few factors that probably influence the query solving process. Little research has been carried out on this topic so far, so we cannot ignore the possibility that other factors are also important for the usability of XML query languages. Therefore, we will first try to find qualitative data that support our assumptions with a thinking aloud experiment. A second goal of this experiment is to find possible other factors that influence the usability of XML query languages. The focus of the first experiment is on language aspects only. Therefore, variables like user experience and data structure are not manipulated in this experiment. This experiment is discussed in Chapter 4, *Experiment 1: Exploring the Model*.

In the second experiment we will try to validate the factors, or a subset of the factors, that are identified in the first experiment. Instead of finding qualitative data with a small set of subjects, we try to find quantitative data with a large group of subjects. In addition, we will also consider user background in this experiment. This experiment is discussed in Chapter 5, *Experiment 2: Validation*.

In the third experiment we will investigate the influence of factors that have not been validated properly yet, for instance because more factors were found in the first experiment than could be validated in the second experiment. In addition, we want to investigate the influence of different XML document types on the performance with different XML languages. This experiment is discussed in Chapter 6, *Experiment 3: Explanation*.

Chapter 4. Experiment 1: Exploring the Model

4.1. Introduction

In this chapter[1] we explore the model of the query solving process that is formulated in the previous chapter. A thinking aloud experiment is used to find indications if our hypotheses about the usability of XML query language are correct and to discover possible new factors that influence the query solving process.

4.2. Research Questions

In this chapter, we want to find answers to the following questions and hypotheses:

1. What are the differences in user performance between XQuery, SQL/XML and XSLT?
2. What are the reasons of performance differences? We will try to see whether the findings of earlier research apply to current XML query languages. In addition, we will try to find support for opinions that are stated in the XML community. Based on these findings and opinions, we formulated the following hypotheses:
   a. Procedural languages result in higher performance compared to less procedural languages because the way humans solve problems is better facilitated by a procedural language. Therefore, we expect that XQuery and SQL/XML performance will be better compared to XSLT. The difference between the languages will be noticeable in the number of activities, transitions, and paths related to constructing the deep structure and constructing the surface structure of a query.
   b. A query language that has consistent names and semantics for its operators, keywords and functions enables users to perform better than with a query language that has inconsistent names and semantics. Based on this, we expect that SQL/XML performance is lower compared to XQuery and XSLT. The differences between the languages will be noticeable by the number of activities and transitions and paths between activities that are related to the construction of the surface structure of a query.
   c. A verbose query language like XSLT is more difficult to use compared to more compact query languages like XQuery and SQL/XML. The differences will be noticeable in a similar way as for consistency of a language. Namely, the differences between the languages will be noticeable by number of

[1] This chapter is based on [Gra04]

activities and transitions and paths between activities that are related to the construction of the surface structure of a query. The difference between SQL/XML and XQuery is smaller, but we consider SQL/XML more verbose because of the extension functions that are needed to translate between relational data and XML data.

d. Users that are more familiar with a particular language type will perform better with corresponding languages. The subjects in this experiment are highly experienced database and SQL users. Based on their expertise with database systems and SQL, we expect that their performance will be the highest with SQL/XML, followed by XQuery, and will be the lowest with XSLT. These differences will be noticeable in all subactivities of the query solving process.

e. Differences between query languages will become more clear with more complex queries compared to less complex queries. Also, performance on simple queries will generally be higher compared to complex queries. The differences between query type will be noticeable in all activities during the query solving process.

3. Are there other possible factors that influence the usability of XML query languages?
4. Which XML query language results in the highest user satisfaction?

Note that we do not include different XML document schemas in this experiment.

Some of the hypotheses that are mentioned here have an opposite direction. For instance, if we consider the user's background we expect that SQL/XML performance will be better compared to XQuery but if we consider the consistency of a language, we expect that XQuery performance will be better than with SQL/XML. The goal of this experiment is to find clues for the correctness of these hypotheses. We will look at performance differences as well as at differences in activities during the query solving process and based on these differences we will discuss which hypotheses are likely to be correct and which hypotheses are less likely to be correct.

The three languages share a common sublanguage to locate and select XML data: Xpath. We expect therefore that the languages behave in a similar way regarding selection of data. For this reason, we check for differences between the number of Xpath errors and the Non-Xpath errors in a language. We expect that the number of Xpath related errors will be globally the same, but it is likely that the non-Xpath related errors between the three languages vary because of the reasons mentioned above.

The table below provides an overview of the activities, transitions, and paths and their associated factors:

Table 4.1. Overview activities and factors

Activity/transition/path	Factor
Total number of activities	Query language, query complexity, user experience
Constructing the deep structure	Procedurality
Understand conditions of search	Procedurality
Constructing the surface structure	Verboseness, consistency, procedurality
Construct deep structure > construct surface structure	Procedurality
Construct surface structure > construct deep structure	Procedurality
Construct surface structure > submit > evaluate result > correction effort or test expression > construct deep structure	Procedurality
Construct surface structure > submit > evaluate result > correction effort or test expression > construct surface structure	Verboseness, consistency
Correction efforts - deep structure	Procedurality
Correction efforts - surface structure	Procedurality
Correction efforts - lexical problems, incorrect expressions, clarify result	Verboseness, consistency
Errors - non-Xpath related	Query language
Errors - Xpath related	Sublanguage Xpath

4.3. Method

4.3.1. Subjects

The subjects involved in the experiment were six male staff members or ex-members of the Large Distributed Databases Group of the Institute for Information and Computing Sciences, Utrecht University. All subjects were highly experienced relational database and SQL users. Experience with XML, XSLT, Xpath and XQuery was measured with a questionnaire (5 point scale questions) and was at beginners or intermediate level. We have chosen to use experienced subjects because the XML query languages are not intended for naïve users and we expected that it would be more easy for experienced users to learn the basics of two new XML query language. The subjects were available for eight hours, divided over two days and participation was on a voluntary basis.

4.3.2. Materials

4.3.2.1. XML Query Languages

For XSLT and XQuery, the same query processor was used: Saxon 7.8 [Saxon]. The subjects used a simple text editor to edit the queries. For SQL/XML a commercial relational database implementation with an interface that was similar to the simple text editor was used.

4.3.2.2. Query Tasks

We use the same classification of query complexity as used in the studies of Reisner [RBC75], Welty & Stemple [WS81], and Schlager [Schl91]:

1. Less complex, or easy, queries are mapping queries with one or more search conditions. An example of an easy query is: *Return the title of the article that has matching id attribute value "1".*
2. More complex, or 'difficult', queries are join queries and grouping queries:
    1. *Join queries:* find nodes based on the content of other nodes, for instance to bring together information from different data sources. The related nodes can be located within the same document or in other documents. An example of a join query is: *List the titles of articles cited by an article that has matching id attribute value "2".*
    2. *Grouping queries:* find nodes that have a common value and perform operations on these nodes. This type of queries is also called 'inverting the hierarchy'. An example of a grouping query is: *Group articles by date and calculate the total number of articles in each group.*

We preferred to use an existing collection of queries and documents. Several benchmarks are developed to test the performance of XML query engines. Examples of XML benchmarks are Xmark [SWK+02], The Michigan benchmark [RPJ+03], X007 [BLL+01], and the Xbench family of benchmarks [YOK02]. The former three benchmarks have the disadvantage that the documents used are not very well suitable for human interpretation and they focus mainly on data-oriented XML applications. We choose to use queries and document structures from the Xbench benchmark. This benchmark, or rather collection of benchmarks, covers several XML applications and types of XML data and is therefore more representative for real life XML use than the other benchmarks. Xbench uses both text-oriented and data-oriented XML applications for both single documents and multiple documents. In addition, the queries that are defined in Xbench “cover the functionality of XQuery as captured in the XML Query Use Cases” [YOK02]. We were also able to find instances of queries classified according to the classification scheme mentioned above. For our experiment, we base our documents and queries on the Xbench Text Centric - Multiple Documents (TC/MD). We decided to use the TC/MD collection because of the following reasons.

First, the structure of the TC/MD is fairly easy to understand. No real specialized knowledge is required to understand the documents in this collection. This is for example not true for the TC/Single Document collection. Second, if the TC/MD documents are useful in this experiment, we can easily include other applications defined in one of the other Xbench benchmarks in future experiments. It is in particular useful to distinguish between data and document-oriented XML applications, without switching to another benchmark. Another argument was that we felt that XSLT was disadvantaged because we only included database experts in this experiment. We did not want to disadvantage XSLT any further by using a data-oriented XML application in this experiment.

4.3.3. Procedure

For each subject, the experiment consisted of two sessions. In the first session the subjects learned and performed tasks with SQL/XML. For the second session the subjects were divided into two groups. One group learned and performed tasks with XSLT and the other group did the same with XQuery. The subjects were asked to solve a set of five query tasks and to think aloud while they were doing this. In general, the think aloud method can be used when information is needed about the mental processes of users that perform complex tasks. Mental processes are not directly observable but subjects can be asked to think aloud, that is to verbalize what they are thinking, when they are performing tasks. The think aloud method can be used to gain information about mental processes of users. The utterances and other observable behavior of the subjects are registered in verbal protocols that represent the mental processes and behavior of the subjects. See [BS96] and [BR00] for a discussion about the applications, validity, and techniques of the think aloud method.

Subject behavior was recorded with a screen capture tool, a microphone, and a video camera that was positioned so that the desk and the subject could be seen. In this manner, we could see what the subject did, what he was reading (the screen or which reference manual), and hear what he said during the think aloud phase of the experiment.

Each session had the following structure:

1. Learning phase of approximately 1.5 - 2 hours. In this phase, subjects learned a query language by studying a tutorial. They were allowed to study the tutorial at home the day before the experiment. The structure of the tutorials is based on [Robie04] and is given below. Part 1-3 was almost identical for each language:
    1. Introduction
    2. Example XML files
    3. Xpath expressions
    4. Query languages specific constructions
    5. Eliminating Duplicate Nodes

6. Joins
7. Inverting Hierarchies
8. Comparison Operators
9. Built-in Functions

   All tutorials had the same size and covered the same problems. The SQL/XML tutorial was slightly shorter, because subjects also received a basic XML tutorial with the SQL/XML tutorial.

2. Introduction to the lab and practice with the query language interface for 20 minutes. Subjects were also introduced to the think aloud method. They were shown a video fragment of a person that performed a think aloud task. It was emphasized that they should not worry about what they were saying, but just trying to verbalize everything that they thought of during the experiment.
3. Think Aloud Tasks of approximately 1 hour. In this phase, the subjects received a set of five query tasks that were related to an XML data set. An example of a query task is: *Find the title of the article of the author named "Ben Yang"*.
4. Subjects filled in an evaluation questionnaire after solving the queries.

The behavior and the verbal utterances of the querying subjects on the videos were scored by the experimenter according to the model of the query solving process (see also Section 3.2, “Model of the Query Solving Process”). We also scored whether the incorrect query included an erroneous Xpath expression or not.

By carefully scoring the subject's utterances and behavior we obtained detailed log files of the subject's query solving process for each query task. Below we demonstrate how we scored subject behavior. The total duration of the query process in seconds is provided between square brackets, a transcript of the subject's behavior and utterances is given, and the corresponding subactivity from the query solving process is provided in italics. In the following example a subject is solving the second query task with language XQuery. The subject uses the query expression that he formulated for the first query task. This was the following expression:

```
doc("collection.xml")/collection/article[@id="1"]/prolog/title
```

Table 4.2. Scoring subject behavior

[Time]	Transcript of subject behavior and utterances
	Scored activity
[0.00]	null
[8.96]	(Start)
[9.28]	Subject reads instructions aloud: "Return the titles of the articles that have a matching language attribute nl"
	Understanding Instructions, Reading
[16.04]	subjects looks at the screen and starts to change the query expression from the previous query task: doc("collection.xml")/collection/ article[@lang='nl']/prolog/title]
	Construct Surface Structure, Edit query
[23.16]	subjects says "ok, and language is nl"
	Construct Deep Structure, Understanding document structure, formulate assumption
[26.36]	subject says "let's ... probably more results .. let's put things in clearly"
	Correction Effort, Clarify result, Non Xpath problem
[28.80]	subject changes query expression to: <resu;ts doc("collection.xml")/collection /article[language="]/prolog/title]
	Construct Surface Structure, Edit query
[29.16]	subject: "hmmm" and changes "resu;ts" in "results"
	Correction Effort, Lexical problem, Non Xpath problem
[30.28]	subject completes expression to: <results> {doc("collection.xml")/collection/article [language="]/prolog/title]} </results>
	Construct Surface Structure, Edit query
[38.40]	Subject submits query to the processor
	Submit query
[44.56]	Subject reads processor message: output completed, reload result query? subject chooses "yes"
	Evaluate Result, Reading processor messages
[46.08]	Subject reads result document
	Evaluate Result, Reading Result Document

[Time]	**Transcript of subject behavior and utterances**
	Scored activity
[46.72]	Subjects says "there seem to be seven of them"
	Evaluate Result, Evaluate Result Document
[50.76]	Subject says "let's see if that's correct"
	Test Expression, Incorrect Deep Structure, Non Xpath problem
[53.88]	Subject browses through the source XML document
	Construct Deep Structure, Understanding Document Structure, Check XML source
[55.84]	Subject says "I'm checking if there's a foreign one that's not included"
	Construct Deep Structure, Understanding document structure, formulate assumption
[59.32]	Subject browses through the source XML document
	Construct Deep Structure, Understanding Document Structure, Check XML source
[70.52]	Subject reads the instructions aloud "return the titles of the articles..."
	Understanding Instructions, reading
[73.64]	Subject browses through the source XML document and says "ok, that one has got one ... that one has got one too ... that one too ... that one too ... that one too ... that one too .."
	Construct Deep Structure, Understanding Document Structure, Check XML source
[106.56]	Subject says "hmmm, they all seem to have got one ... so"
	Construct Deep Structure, Understanding Document Structure, Establish meaning
[111.68]	Subject reads instructions aloud: "Return the titles of the articles that have a matching language attribute nl"
	Understanding Instructions, reading
[120.40]	Subject says "Yes, it seems to be all right, so I'm done with this one"
	Evaluate Result, Evaluate Result Document
[123.12]	(Stop)

This particular query solving process consists of 20 subactivities (the null, start, and stop marks are not a part of the query solving process) and lasted 114 seconds (123.12 - 8.96 seconds). For each query solving process the number of subactivities, and the number of relevant transitions between activities, were counted automatically. To gain

insight of the reliability of our scoring method, we let a student assistant score two query tasks. In addition, we scored some of the query tasks again after two weeks. We compared the resulting log files of the same tasks and found no large differences between the different versions. Therefore, we conclude that the method of scoring is sufficiently reliable.

The correctness of the subject's result queries were rated as correct or incorrect by the experimenter. We discuss the results of the experiment in the following section.

4.4. Results

The performance of the subjects was measured in two ways: First, in terms of effectiveness, that is the correctness of the query expression formulated by the subject. The correctness of the query was determined by the experimenter and an expression was simply rated as correct or incorrect. Second, in terms of efficiency; in this experiment efficiency is measured as the number of activities and transitions in the model. These were counted by the experimenter on the basis of the verbal protocols and the video recordings. The fewer the number of activities and transitions, the more efficient the performance. Third, to get a first indication of the satisfaction of the query languages, we asked subjects for their opinions on the three query languages after they participated in the experiment.

4.4.1. Correctness of the Queries

The following table shows the percentages and the absolute numbers of the correct and incorrect results between parentheses for simple and difficult queries for the three query languages. Each subject worked with SQL/XML and with either XQuery or XSLT and solved five query tasks with each language. The five query tasks consisted of three simple and two difficult query tasks. Six subjects participated in the experiment, so the total number of query tasks is as follows:

1. SQL/XML: 30 query tasks, more specifically:
    a. 18 simple query tasks
    b. 12 difficult query tasks
2. For both XQuery and XSLT: 15 query tasks, more specifically:
    a. 9 simple query tasks
    b. 6 difficult query tasks

Table 4.3. Correctness of Simple and Complex Queries.

	SQL/XML	XQuery	XSLT
Simple	94% (17)	78% (7)	100% (9)
Difficult	42% (5)	100% (6)	67% (4)
Total	73% (22)	87% (13)	87% (13)

The data in this table shows the following. First, SQL/XML has lowest percentage correct queries (73%) and seems therefore to be the least effective. Second, the effectiveness rates with XSLT and XQuery are equal (87%) in this experiment. Third, with XQuery 78% of the simple queries where solved correctly, and all difficult queries were correctly solved. With XSLT, all simple queries were solved correctly, and 67% percent of the difficult queries were solved correctly. So, although the overall effectiveness with XQuery and XSLT was equal, there were differences in effectiveness for the different query types.

We noticed two types of erroneous result queries in this experiment:

1. Surface structure errors. This type of error occurred when a subject knew which data should be retrieved from the document collection, but was not able to formulate the correct query. This occurred with seven difficult queries, (5x SQL/XML and 2x XSLT). One subject made this type of error with all difficult queries (2x SQL/XML, 2x XSLT).
2. Deep structure errors. We noticed two reasons for deep structure errors:
   a. The subject misunderstood the instructions and formulated a query that produced a result that did not match the instructions. This occurred three times and only with simple queries (2x XQuery and 1x SQL/XML). In these cases, the subjects were a bit hasty and did some sloppy reading. We do not consider this as a serious type of error because we think that the subjects could do better if they were more focused.
   b. During the query solving process, the subjects forgot the initial conditions of search. This occurred two times and only with difficult queries. In these cases the subjects were so focused on solving a subproblem, that they forgot the overall problem and they stopped when they found the correct solution for the most difficult subproblem. This happened two times, both with difficult SQL/XML queries.

We summarize the results in this experiment regarding correctness as follows. Most erroneous queries were made with SQL/XML. The correctness of the queries produced with XSLT and XQuery are equal in this experiment, but we consider the errors made with XSLT more serious than the errors made with XQuery. Therefore, XSLT seems to be less effective than XQuery.

4.4.2. Efficiency

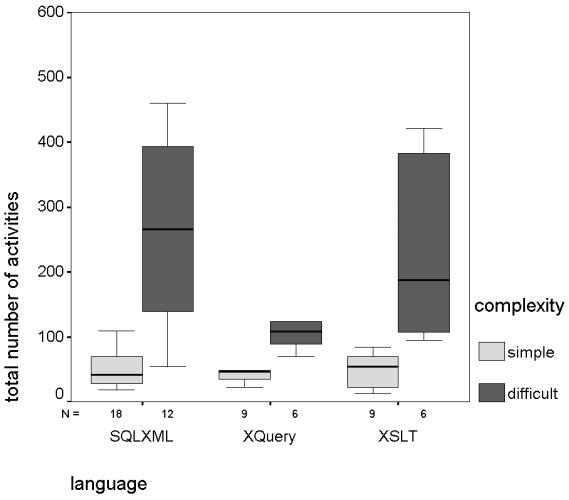
According to the ISO standard "Guidance on Usability" efficiency is described as follows: the resources that are utilized in relation to accuracy and completeness through which users achieve certain results. See also Section 2.2, “Usability Aspects”. A common measurement for efficiency is the time that is needed to complete certain tasks. However, in this experiment we have a more refined measurement available for efficiency, namely the number of the subject's activities and transitions between activities during the query solving process. We can extract this information from the log files of the think aloud protocols. The number of activities and transitions between activities of the subjects are an indication of the resources utilized by the user to achieve a certain result. We will also discuss the relation between time spent on the tasks and the number of subactivities during the query solving process. In this section we present the differences of efficiency of the query solving process for SQL/XML, XQuery, and XSLT for simple and difficult queries.

4.4.2.1. Total Number of Activities

We provide a boxplot with the total number of activities for all queries for SQL/XML, XQuery, and XSLT. The number of activities on the simple queries and difficult queries are presented separately. The boxplots show the median, quartiles, and extreme values. They represent the interquartile range which contains the 50% of values. The whiskers are lines that extend from the box to the highest and lowest values. Extreme values were excluded from the boxplots. A line across the box indicates the median. The values of N in the diagram represent the number of queries that are included in the boxplot. For instance, the first boxplot has a N value of 18 and this means that the boxplot of the simple SQL/XML queries consists of 18 queries. The boxplots indicates that from the 18 queries the total number of activities ranged from approximately 15 to 105.

The total number of activities is a measurement for the efficiency of use with a query language and is the sum of all activities during the query solving process of one query task. The example we provided in Table 4.2, “Scoring subject behavior” consisted of 20 activities and this number is included in the boxplot of the simple query tasks solved with XQuery. See the figure below:

Figure 4.1. Total Number of Activities



We notice three striking differences between the total number of activities. First, subjects needed less subactivities for simple queries compared to difficult queries for all languages during the query solving process. Second, the total number of activities is much smaller for XQuery compared to both XSLT and SQL/XML. The difference between SQL/XML and XSLT is smaller, but subjects who used XSLT tended to need less subactivities. Third, the differences between the query languages become larger with more complex queries. The differences between SQL/XML, XSLT, and XQuery are very small for the simple queries, but the number of activities on difficult queries with XQuery is much lower compared to SQL/XML and XSLT.

We consider the number of the subject's activities during the query solving process as an indication for how efficiently the query languages are used on these queries. A more common measurement for efficiency is the time that is spent on the queries. We registered how much time the subject's needed for each query and we tested whether the time spent on a query was correlated with the total number of activities during the query solving process. We computed Pearson's r and found a significant positive correlation between time spent on the query and number of activities during the query solving process ($r = .98$, $n = 60$, $p < .0005$, two-tailed). Therefore, the number of subject's activities is strongly correlated to the time spent on query tasks and can indeed be used as an indicator for efficiency. In addition, the number and nature of specific subactivities can be used to detect reasons for efficiency differences between query languages.

During the experiment, we had the impression that the user effort for the simple queries was fairly constant between the different simple type queries. However, for difficult

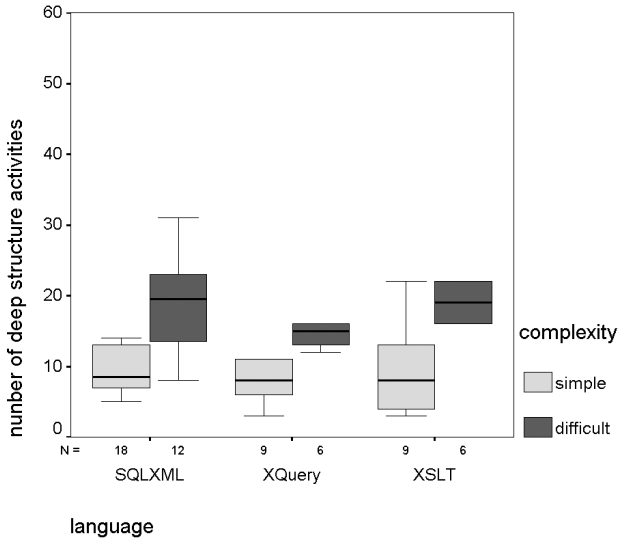
queries we had the impression that the two queries were not equally difficult for all languages. Therefore, we compared the total number of subactivities per language for the two difficult query types: the join and the grouping query. See Figure A.2, “Total Number of Activities - Join and Grouping” in Appendix A, *Additional figures*. This figure shows that for SQL/XML, join and grouping queries are solved equally efficient in this experiment. For XQuery, we notice that grouping queries are solved less efficient than join queries. For XSLT we notice the opposite: grouping queries are solved more efficient than join queries. We assume that this is related to the specialized construct for grouping in XSLT 2.0 and the lack of such a similar operator in XQuery.

4.4.2.2. Constructing the Deep Structure

In Section 3.2.4, “Construct Deep Structure” we described the activity during the query solving process where the user constructs the deep structure of a query. This activity consists of two main subactivities that also consist of several subactivities: Understand Conditions of Search and Understand Document Structure. The number of deep structure activities that are presented below consist of the total number of these subactivities. An example of a deep structure subactivity is *establish the meaning of the document structure* and this subactivity was logged when a subject remarked, for instance, "Hmmm, and a title is a child of article...". An example of *understand conditions of search, decompose problem* is when a subjects says: "Ok, so I first need to find all author elements, and than I need to check whether they have the following name. Oh, and I need to return their articles as well".

The number of deep structure activities for the three query languages and for the simple and difficult queries are presented in the following figure:

Figure 4.2. Number of Construct Deep Structure Activities



This figure shows that the number of construct deep structure activities for simple queries is approximately equal for all languages. For the difficult queries, the data shows that XQuery has led to a lower number of deep structure activities compared to both XSLT and SQL/XML.

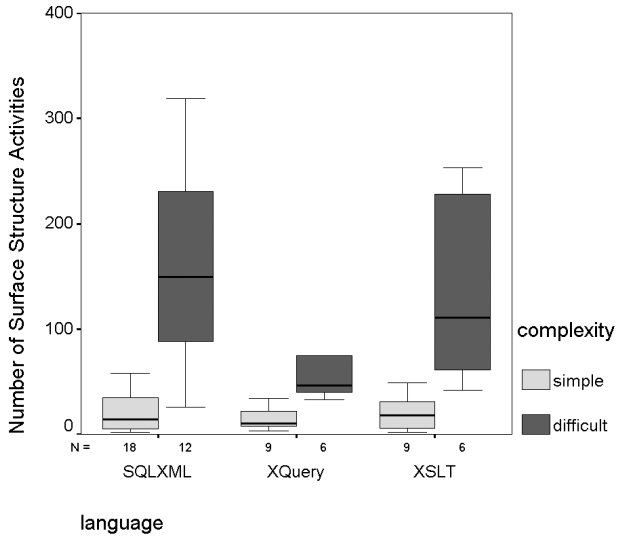
In a similar way we looked at the number of conditions of search activities, but we observed no differences between the query languages for the number of understanding condition of search activities that the subjects needed. See also Figure A.1, “Understand Conditions of Search Activities” in Appendix A, *Additional figures*. These results and the implications for the assumed effect of procedurality are discussed in Section 4.5, “Discussion”.

4.4.2.3. Constructing the Surface Structure

In Section 3.2.5, “Construct Surface Structure” we described the activity during the query solving process where the user constructs the surface structure of a query. Construct surface structure activities are scored, for instance, when a subject says: "I think I need a count function or something". This is scored as *formulate assumption about suitable expression*. When a subjects starts to search for such an expression in the tutorial this is scored as *check language reference*. When a subject is typing or cutting and pasting query language expressions, this is scored as *edit expression*.

We show the plots of the number of construct the surface structure activities below:

Figure 4.3. Number of Surface Structure Activities



We noticed no differences for the simple queries between the languages. We also noticed that the subjects generally needed a lower number of construct surface structure activities for the simple queries than for the difficult queries. More interestingly, for the difficult queries we noticed a large difference between the number of construct surface structure activities between SQL/XML, XSLT, and XQuery. Subjects who worked with XQuery needed the fewest construct surface structure activities. The difference between SQL/XML and XSLT was smaller, but subjects who worked with XSLT needed fewer construct surface structure activities than subjects who worked with SQL/XML.

4.4.2.4. Transitions

A transition is a succession of two activities and a path is a succession of more than two activities. For instance, when a subject is editing a query (scored as a subactivity of construct surface structure) and remarks "Wait a minute, is the last name of a person coded as a lastname or as a surname element?" (scored as formulate assumption about document structure) than this is counted as a transition from Construct Surface Structure to Construct Deep Structure. We counted the following transitions and paths:

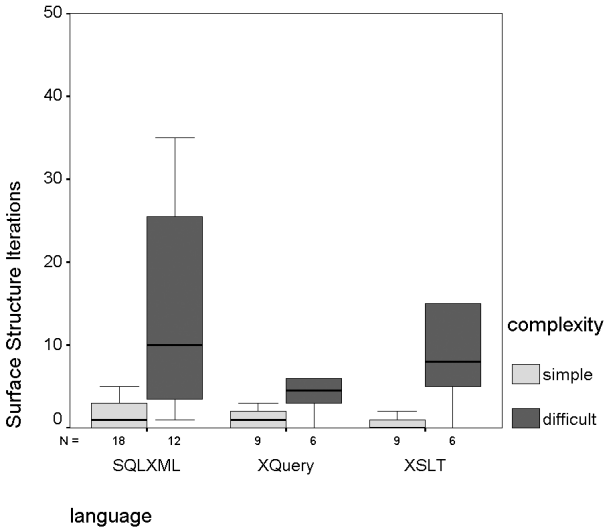
1. Construct Deep Structure > Construct Surface Structure
2. Construct Surface structure > Construct Deep structure
3. Construct Surface Structure > Submit > Evaluate Result > Correction Effort or Test Expression > Construct Deep Structure

4. Construct Surface Structure > Submit > Evaluate Result > Correction Effort or Test Expression > Construct Surface Structure

The number of the first three transitions and paths are presented in Figure A.3, “From deep structure to surface structure”, Figure A.4, “From Surface Structure to Deep Structure”, and Figure A.5, “Constructing and Debugging the Surface Structure” respectively. The differences between the languages and query types on these transitions and paths are negligible. We will discuss these results in Section 4.5, “Discussion”.

The number of the last path is presented below:

Figure 4.4. Surface Structure Iterations



This figure shows that there are noticeable differences between the languages and the query types. For simple queries, the number of transitions is nearly equal for all languages. For difficult queries, we noticed that the number of transitions is the largest for SQL/XML, somewhat smaller for XSLT, and the smallest for XQuery.

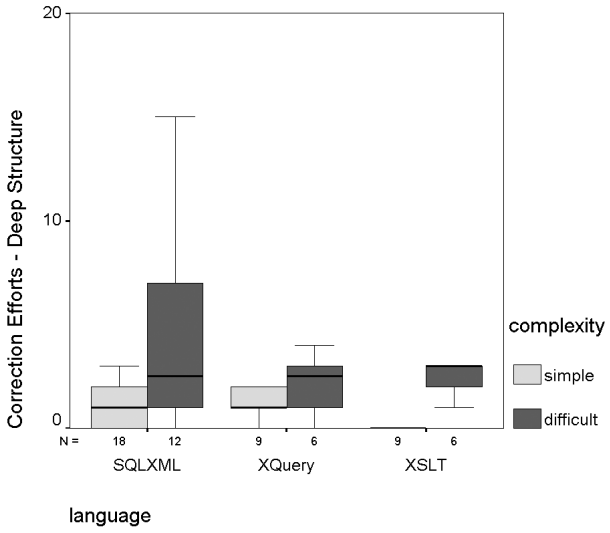
4.4.2.5. Correction Efforts

Correction efforts are attempts of the user to correct a query expression, see also Figure 3.1, “Query Solving Process: Top Level”. The user can correct errors with a correction effort during the formulation of the surface structure or after he submitted the query and evaluated the result. An example of the former is when a user is typing a query expression but notices that he made a mistake. An example of the latter is when a user has submitted a query and notices that the query expression is syntactically

not correct. An important aspect of the correction effort activity is that the error that is corrected is determined by the user, but it is possible that the correction attempt does not solve the real problem in the query expression. We assume that various types of correction efforts are related to different aspects of the query languages. We discuss these types in this section.

Corrections of the Deep Structure. This type of correction effort was made when a query was syntactically correct, but included a semantic problem. The subject realizes that the result of the query was not according to the original query task description. This type of corrections were scored when a subject remarks, for instance, "Ok, this is correct. No, did I only need to return the lastnames of the persons?". The following figure shows the number of correction efforts that were related to problems with the deep structure of the query:

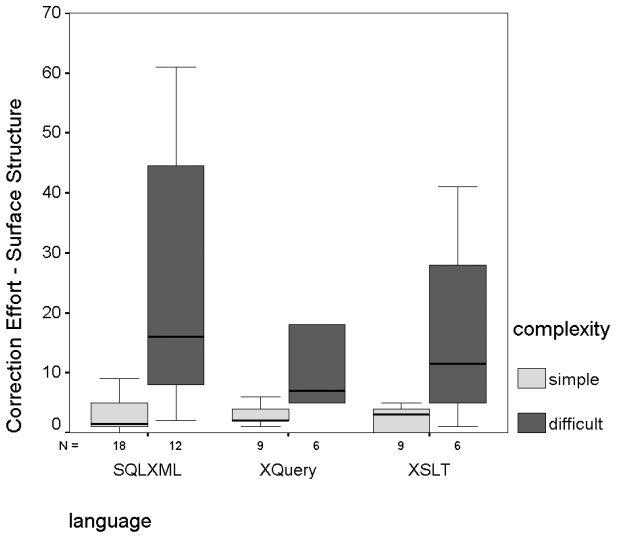
Figure 4.5. Correction Efforts - Deep Structure



Few correction efforts related to the deep structure were made for the simple queries. The differences between the languages on simple queries are also very small. On difficult queries we observed similar results. The total number of deep structure correction efforts is slightly higher compared to the simple queries and the differences between the languages are small. However, subjects working with SQL/XML tended to make more deep structure correction efforts compared to XQuery and XSLT. The difference between the latter is negligible. We will discuss these results further in Section 4.5, “Discussion”.

Corrections of the Surface Structure. Surface structure errors are also expressions with a semantic problem, namely the result of a query is not according to the deep structure of the query. This type of correction effort was scored when a subject remarks, for instance, "I want to see two titles here, where are my titles?". The following figure shows the number of correction efforts related to the surface structure of a query.

Figure 4.6. Correction Efforts - Surface Structure

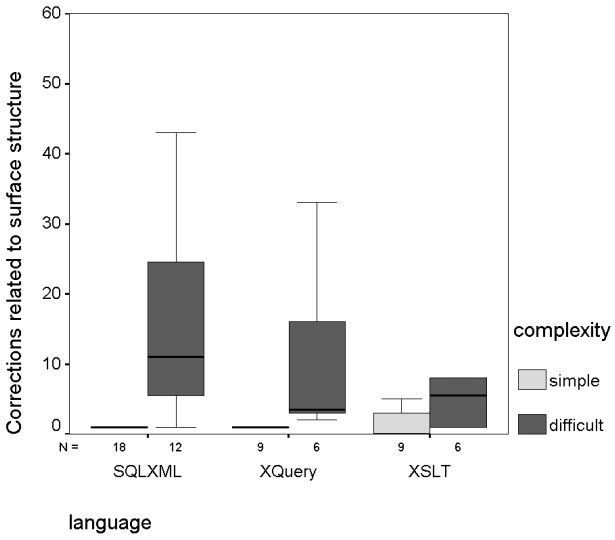


We noticed similar trends as with the number of construct surface structure activities. Again, no differences for the simple queries between the languages were noticed. Also, fewer correction efforts related to the surface structure were made with the simple queries compared to the difficult queries. Finally, the differences between the query languages for the difficult queries were as follows. The fewest surface structure related correction efforts were made with XQuery, followed by XSLT. With SQL/XML the most surface structure related correction efforts were made. We discuss these results in Section 4.5, “Discussion”.

Corrections Related to the Surface Structure. In our model of the query solving process, we defined three types of errors or problems that are related to the formulation of the surface structure of the query. Namely, lexical problems, incorrect expressions, and results that need to be clarified. A query with a lexical problem or with an incorrect expression is syntactically incorrect. Correction efforts of lexical problems were scored when a subject remarked, for instance, "That comma needs to be replaced, obviously". When a subject continues with "Hmmm, the count function is probably not used correctly after all" this was scored as a correction effort of an incorrect expression. Correction efforts to clarify a result are scored when a subject says something like

"Let's make it a bit more readable". The following figures show the summarized values for all correction efforts and test expression activities that were related to lexical problems, incorrect expressions, and to clarify the result query:

Figure 4.7. Correction Efforts related to the Surface Structure



This figure shows that for the difficult queries the number of corrections related to the surface structure (lexical problems, incorrect expressions, clarify result) is the highest with SQL/XML compared to both XQuery and XSLT. The difference between XQuery and XSLT is less clear. For XQuery, the range of these problems is somewhat larger than for XSLT and also higher values are noticed, but the median is slightly lower than for XSLT. Therefore, the difference between XQuery and XSLT is small, but subjects who used XQuery in this experiment tended to encounter less surface structure related problems compared to subjects who used XSLT. For the simple queries, the number of correction efforts is very low, and no clear differences can be observed between the languages.

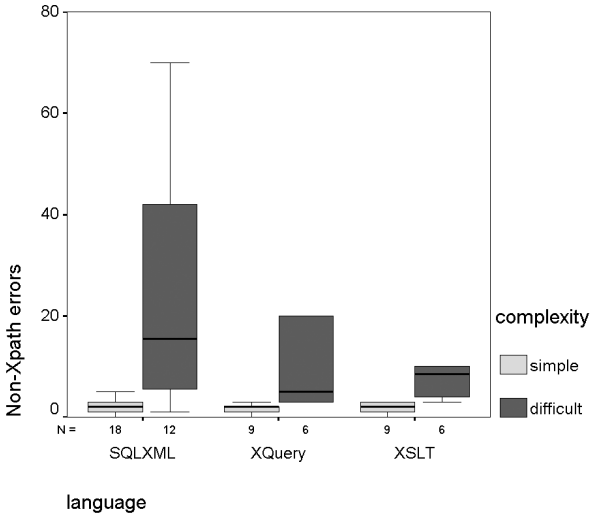
4.4.2.6. Errors

In this section we present the number of errors made by subjects during the query solving process. The difference with correction efforts is that the type of error, Xpath related or non-Xpath related, is determined by the experimenter instead of the subjects. We distinguished between Xpath and non-Xpath related errors because we expected that the number of Xpath errors would be approximately the same for all languages.

We start with the errors that were not related to Xpath and continue with the Xpath related errors.

The following figure represents the number of errors made by subjects that were not related to Xpath:

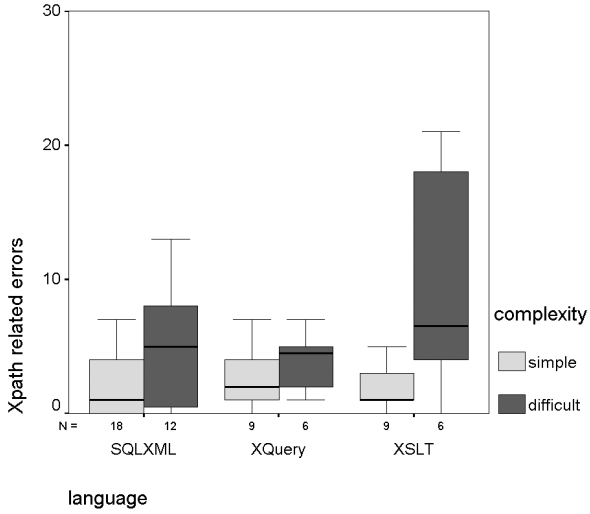
Figure 4.8. Errors - Non-Xpath related



For the difficult queries we can see the following in this figure. In this experiment the largest number of errors during the query solving process was made with SQL/XML. There is no clear difference between the number of errors made with XSLT and XQuery. For simple queries there is no difference between the three languages, and the number of errors is higher with more difficult queries compared to the simple queries.

The following figure represents the number of Xpath related errors:

Figure 4.9. Errors - Xpath related



Most important observation is that the number of Xpath errors is the highest with XSLT. The difference between SQL/XML and XQuery is not obvious. Difficult queries lead to more Xpath errors compared to simple queries, but the difference is not very large.

4.4.3. Satisfaction

The subjects completed a questionnaire about the query languages after the think aloud tasks. We asked them to name three positive and three negative aspects of the query language used in the experiment. We summarize the answers below. All subjects mentioned the ease of use of Xpath expressions, so we don't repeat this comment for every language.

Most important positive remarks about SQL/XML is the use of the SQL framework. Five of six subjects mentioned this. One subject also commented that the use of XMLelement() was nice, but no further positive comments were made. Some very general negative comments were made, for instance that SQL/XML is not intuitive, verbose, and expressions are complex to formulate. More specific were comments on the data model: the use of both SQL table and XML was considered to be confusing (three subjects made this comment) and, in the same line, the distinction between Xpath strings and SQL strings was difficult (two subjects made this comment). One subject noted that conditions could be made in the WHERE clause, as well as in Xpath expressions. He thought this was redundant and confusing.

Subjects mentioned the following positive aspects for XSLT. Creating new XML structures is easy with XSLT. A somewhat similar remark was that the structure of the query reflects the structure of the result and that for this reason debugging was fairly easy. One subject liked the automatic type casts and set - atomic value mapping. Negative remarks were related to the verboseness of XSLT: XML syntax was not considered to be a good syntax type for a query language. Also, the default templates were considered cumbersome. More individual remarks were that the xsl:for-each-group element was considered to be counterintuitive, that it is hard to get back to nodes higher in the XML source tree, and that the use of both 'eq' and '=' is confusing.

Positive remarks about XQuery were that the language and the FLOWR expressions were simple and intuitive. Two subjects mentioned that, in explicit contrast with SQL/XML, the use of one data model (XML) was much easier than the other two data models (relational and XML). Individual remarks were made about the nesting of queries, variable bindings and automatic type casts. Interestingly, some negative remarks about XQuery were related to familiarity to SQL. One subject made the general notion that there is sometimes conflicting SQL knowledge. For instance in a language that is similar to SQL, one would expect a Group By operator. The lack of standard constructs like 'Group By' was also mentioned by another subject. In addition, some individual remarks were made: the difference between For and Let was not really clear. XQuery expressions can become complicated very fast, and than it is difficult to understand what exactly is going on (note that this is somewhat contradicting the first general remarks). Like in XSLT, the eq versus = was considered strange. One subject noted that the structuring of the RETURN clause was confusing. With which he meant that one variable can be returned without further problems, while multiple variables must be embedded within an XML element and curly brackets. In the experiment, he was stuck for half an hour on this particular construct.

In general, the subjects in this experiment had a negative opinion about SQL/XML. The general idea of using the SQL framework for an XML query language was considered to be a good idea, but the current design was considered as counter intuitive, verbose, and complex. The main reason for this, is probably the inclusion of two different data models in one language. The subjects were generally more satisfied with both XSLT and XQuery. This is remarkable, because we assumed that this particular group of highly experienced database users would have a very positive opinion about SQL with additional XML functionality.

4.5. Discussion

In this section we discuss the results of the experiment. The following table summarizes the research questions: it consists of the factors and the accompanying activities, transitions, or paths:

Table 4.4. Summary of the research questions

Factor	Activity/transition/path
Query language	Total number of activities
	Errors - non-Xpath related
Query complexity	Total number of activities
Procedurality	Constructing the deep structure
	Understand conditions of search
	Construct surface structure
	Construct deep structure > construct surface structure
	Construct surface structure > construct deep structure
	Construct surface structure > submit > evaluate result > correction effort or test expression > construct deep structure
	Correction efforts - deep structure
	Correction efforts - surface structure
Consistency	Constructing the surface structure
	Construct surface structure > submit > evaluate result > correction effort or test expression > construct surface structure
	Correction efforts - lexical problems, incorrect expressions, clarify result
Verboseness	Constructing the surface structure
	Construct surface structure > submit > evaluate result > correction effort or test expression > construct surface structure
	Correction efforts - lexical problems, incorrect expressions, clarify result
User experience	Total number of activities
Sublanguage Xpath	Errors - Xpath related

The data in this experiment show that there are differences in effectiveness between SQL/XML, XQuery, and XSLT, in particular for difficult queries. The percentage correctly solved difficult queries is 100% with XQuery, 67% with XSLT and 42% with SQL/XML. For simple queries we noticed that most errors were made with XQuery, but we consider the errors on simple queries as not important because these errors were due to hasty and sloppy reading of the task descriptions. In addition, the

subjects who made these errors were perfectly capable of solving the difficult queries correctly with this language. We are quite sure this type of error has little to do with the query language that is used. For this reason, the number of correctly solved simple queries is for each language nearly 100%.

In this experiment, efficiency is defined as the number of the subject's activities during the query solving process. A more common measurement for efficiency is the time that subject's need to solve a particular task. In addition to determine the number of activities during the query solving process, we also registered the time subjects needed during this process. We noticed that the time and the total number of activities of the query solving process were highly correlated (98%). Therefore, we conclude that the number of activities during the query solving process is a valid and useful measurement of efficiency.

Complexity. We found no noticeable differences for efficiency of the query solving process on simple queries. The number of activities on simple queries is for all languages approximately 50. This means that simple queries are solved with approximately 50 subactivities of the query solving process in all languages. More importantly, for the join and the grouping queries, or the difficult queries, the differences between languages are much bigger. As we can see in Figure 4.1, “Total Number of Activities”, the approximate number of subactivities for SQL/XML is 260, for XSLT 190, and for XQuery 110. In general, this indicates that difficult queries are solved more efficiently with XQuery compared to SQL/XML and XSLT. This indicates that the hypothesis about the influence of query type is correct: in this experiment, the performance differences between the query languages became larger with the more difficult queries. It was already noticed that the correctness for difficult queries is highest for XQuery, followed by XSLT, and the lowest for SQL/XML. Finally, these results suggest that the effectiveness of use of a query language is related to the efficiency of the query solving process for a language. Thus apparently what happens during the query solving process is a reliable indicator of the final query language performance. Based on this, we conclude that we provided an accurate model of this process. We also noticed efficiency differences within query languages with respect to the type of difficult query. With XSLT, grouping queries are solved more efficiently than join queries. With XQuery it is the opposite: grouping queries are solved less efficiently than join queries. In SQL/XML, joins and grouping queries are solved equally (in)efficient. This suggests that with XSLT grouping queries are easier to solve than join queries and the opposite is true for XQuery. The reason for this may be that XSLT 2.0 has a specialized grouping element: for-each-group, while XQuery lacks a specific grouping operator. Therefore, a grouping problems in XSLT is easier to map to a query expression than a join problem, although a grouping problem in itself is more complex than a join problem.

We will now discuss the differences in correctness and efficiency with respect to the research questions. We will only discuss the differences between the languages on difficult queries, because we noticed almost no differences between the languages on

simple queries. The relation with each factor to specific parts of the query solving process are described in Table 4.1, “Overview activities and factors”

Procedurality. First, we assumed that performance with a more procedural language would be better than with a non-procedural language. The assumed reason for this, is that according to Welty and Stemple a procedural problem descriptions suits better with the natural way humans solve problems compared to non-procedural problem descriptions. Therefore, the translation from the deep structure of a query to the surface structure of a query will be easier with a procedural language, e.g. XQuery and SQL/XML, than with a non-procedural language, e.g. XSLT. In terms of the model of the query solving process, this would be noticeable in the following activities and transitions. First, the number of transitions and paths between the construct deep structure activities and construct the surface structure activities. Second, the number of construct deep structure activities, and in particular the subactivities related to understand the conditions of search, and the number of construct surface structure activities. Closely related to these activities are the number of correction efforts - deep structure and correction efforts - surface structure. We expected that there would be a larger number of these activities for XSLT compared to both XQuery and SQL/XML. We noticed that subjects using XSLT needed more construct deep structure activities compared to subjects using XQuery, but not compared to the subjects using SQL/XML. See Section 4.4.2.2, “Constructing the Deep Structure” for the presentation of the results. The number of construct deep structure activities for XSLT and SQL/XML is approximately the same. No differences were noticed for understanding the conditions of search subactivities between either XSLT, SQL/XML, and XQuery. For the number of construct surface structure activities we noticed similar results: lowest number for XQuery and more activities for both XSLT and SQL/XML. See also Figure 4.3, “Number of Surface Structure Activities”. The number of correction efforts related to the deep structure are presented in Figure 4.5, “Correction Efforts - Deep Structure”. We found that subjects working with SQL/XML tended to make more deep structure correction efforts compared to XQuery and XSLT. The difference between XSLT and XQuery is negligible. The fewest correction efforts of the surface structure were made with XQuery, followed by XSLT. With SQL/XML the most surface structure related correction efforts were made. See also Figure 4.6, “Correction Efforts - Surface Structure”. The results for the transitions between the deep structure and the surface structure activities are presented in Section 4.4.2.4, “Transitions”. We found no perceptible differences between the languages for these transitions. We noticed that the number of transitions from constructing the surface structure to the deep structure that included other subactivity was negligible. We will return to this issue when we discuss the value of the model for this experiment.

We assumed that the less procedural XSLT would result in the lowest number of the activities, transitions, and paths mentioned above compared to the more procedural SQL/XML and XQuery. However, this did not occur in any of these activities, transitions, and paths. Contradictory to our expectations, the performance with the more procedural language SQL/XML was in general lower compared to the

non-procedural XSLT. The number of construct deep structure and surface structure activities and their accompanying correction efforts with XSLT is lower compared to XQuery. This may be caused by the differences in procedurality between these languages. However, we consider this as rather implausible because the differences between SQL/XML and XSLT cannot be explained by their difference in procedurality and, more important, there are no differences between the transitions and paths between the surface structure and the deep structure for XSLT, XQuery, and SQL/XML. Therefore, we conclude that the data in this experiment did not support the hypothesis that there is a strong relation between the level of procedurality of a language and the performance with a language. Other factors seem to have a larger influence on the performance with the language. We have two possible explanations for this outcome. First, we noted that the result queries of the subjects using XSLT in this experiment generally consisted of only one or two template rules. Therefore, it is likely that the non-procedural nature of XSLT remained implicit in this experiment. Second, when we zoomed in on more detailed aspects of the query solving process, we noticed that the numbers of transitions and activities became too small to make meaningful interpretations. For instance, the number of transitions from constructing the surface structure to submit, evaluate result, correction efforts or test expressions, and to deep structure, was very low. It is possible these observed numbers were very low due the number of subjects or to limitations of the thinking aloud method, and that we simply missed certain instances of this path. We will return to this issue shortly.

Consistency. Second, we assumed that a consistent language would enable a better performance compared to a less consistent language. This would be most noticeable in the number of construct surface structure activities (see Figure 4.3, “Number of Surface Structure Activities”), correction efforts that are related to the surface structure: lexical problems, incorrect expressions, and clarify result (see Figure 4.7, “Correction Efforts related to the Surface Structure”), and the number surface structure iterations (see Figure 4.4, “Surface Structure Iterations”). Based on this hypothesis, we also expected that performance with SQL/XML would be lower compared to XSLT and XQuery. The data in this experiment support this hypothesis. Performance with SQL/XML is lower compared to both XSLT and XQuery and also the number of construct surface structure activities, correction efforts related to this activity, and the number of surface structure iterations are highest for SQL/XML. In addition, most subjects made negative comments on the inconsistencies and counter intuitive aspects of SQL/XML.

Verboseness. Third, we assumed that a compact language would enable a better performance compared to a verbose language. The effect of verboseness of a language would be noticeable in the same activities, transitions, and paths as for consistency. The data in this experiment partly supports this hypothesis, as XSLT indeed resulted in more construct surface structure activities (see Figure 4.3, “Number of Surface Structure Activities”) and number of surface structure iterations (see Figure 4.4, “Surface Structure Iterations”) compared with XQuery. However, as we have stated above, these transitions and processes are the highest for subjects who used SQL/XML.

This is also true for the related correction efforts activities (see Figure 4.7, “Correction Efforts related to the Surface Structure”): SQL/XML resulted in the highest number of correction efforts and there is no noticeable difference between XSLT and XQuery. Therefore, verboseness is probably not the main reason for performance differences. The consistency of a language is likely to be more important than the procedurality and the verboseness of a language. This also may explain the smaller difference between XSLT and XQuery.

User Experience. Based on the familiarity with the language we expected that subjects would perform best with SQL/XML. Surprisingly, we found that the opposite was true. Our hypothesis was not far-fetched, for instance because the subjects in the experiment commented that they liked the idea of using the SQL framework. However, the inconsistencies in the language and the complexity of the conversion functions make SQL/XML a very difficult language to use. The performance differences between XQuery and XSLT cannot be explained by the SQL background of the subjects. Apart from the remark that XML is not a good syntax choice for a query language, no negative comments were made to XSLT. Comments for XQuery were generally very positive, but the superficial similarity with SQL was one of the negative comments. According to two subjects, knowledge about certain SQL constructs was sometimes conflicting while working with similar XQuery constructs.

Xpath Embedding. In addition to our expectations about query language aspects that influence the usability of a language, we also found indications for factors that we did not think of in advance. We encoded the number of Xpath errors and non-Xpath errors because we expected that the number of Xpath errors would be the same for all languages. We noticed that this was not true: the distribution for differences between non-Xpath problems was the same as for the global correctness and efficiency for the languages. XQuery had the fewest errors, XSLT intermediate, and SQL/XML the most errors. However, for Xpath related errors we noticed another distribution: XSLT had most Xpath errors, followed by SQL/XML and XQuery respectively. This indicates that the method of Xpath embedding in XSLT is somehow more difficult to use than with XQuery and SQL/XML.

Expression Type. Another factor that might influence the usability of a language that we did not expect, is the use of a specialized operator for a particular task. The use of XSLT's for-each-group element seemed to lead to better performance on grouping queries compared to the use compound expressions in XQuery. With expression type we denote the difference between a like XSLT's for-each-group or SQL's group by, and compound expressions like in XQuery where a grouping problem must be expressed with a combination of for and/or let keywords.

Document Structure. Finally, although we have chosen a very simple and clear XML structure and document, we noticed several errors that were due to problems with the document structure. For instance, the title element of an article was not directly located as a child of article, but in a child element of article labeled prolog. Almost

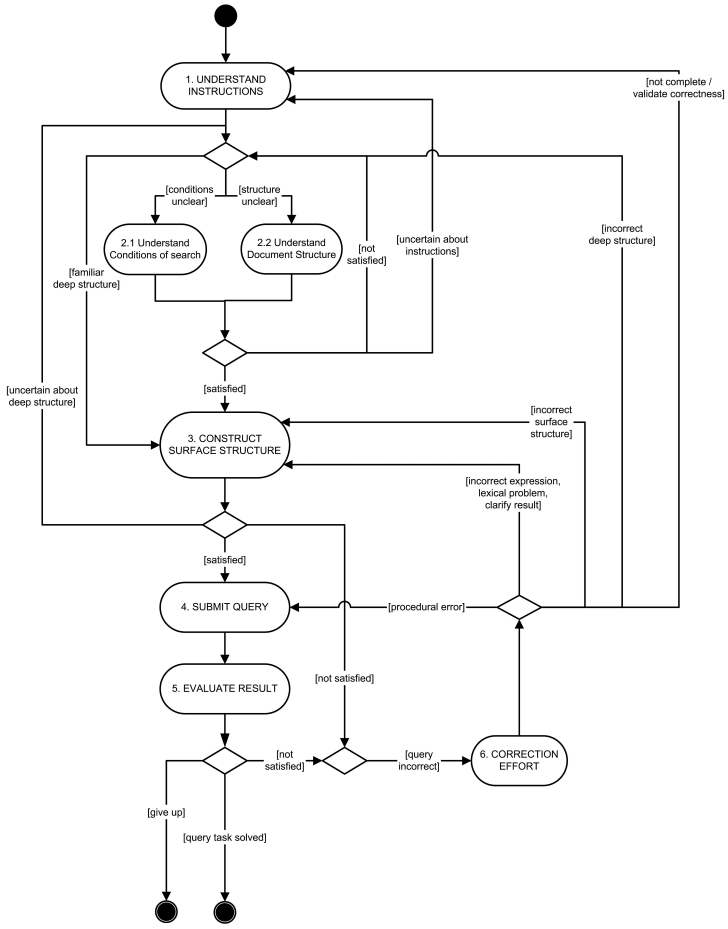
all subjects had problems with this, of course unintended, illogical structure and spent time debugging their queries only because they were unaware of this construct. A few similar problems were noted and this is a pity, because we were mainly interested in how users performed with three XML query language and not how much time it cost to learn illogical XML structures.

Simplification of the Model. In this chapter we explored our assumed model of the query solving process with a thinking aloud experiment. The model is based on previous models, our personal experience as XML query language user, and a pilot test with one subject. In this experiment we noticed that the model was fairly accurate and that we were able to use the model to interpret the behavior and utterances of the subjects. However, we also found some limitations of the model and the method we used. First, we noticed that the distinction between the activities *test expression* and *correction effort* was somewhat artificial. Sometimes subjects made the comment that they were going to test a subquery and found and corrected an error instead. Also, it occurred that subjects thought they made an error but concluded after evaluating the expression that the query was correct after all. Therefore, we concluded that these subactivities could be merged in the model of the query solving process. Second, we had some difficulties with observing specific subactivities. For instance, we noticed that subjects sometimes read the query task aloud and immediately started to type a query. Although not observable, it is impossible that the subject did not make some interpretation of the query task and formulated some ideas about which query expressions are required to solve the query task. With other subactivities we had no difficulties with the observation. For instance, when a subject edited a query, read reference material, or submitted a query. Therefore, we concluded that subactivities that represent mental actions of the subjects were more difficult to observe than visible behavior. This implies that transitions that include multiple subactivities can be interpreted in several ways. For instance, a large number of the following iterations was found:

- Construct Surface Structure > Submit > Evaluate Result > Correction Effort / Test Expression > Construct Surface Structure

See also Figure 4.4, “Surface Structure Iterations”. According to our model, such an iteration represents an attempt to formulate a query expression of a particular problem that is constructed and selected in the construct deep structure activity. However, we noticed that mental behavior of the subjects was sometimes difficult to observe. Therefore, some cautiousness is advised at interpreting the transitions between subactivities. We have more confidence in meaning of number of subactivities than in the number of transitions between subactivities. Third, the model of the query solving process that we defined in the previous chapter was more detailed than required to explore our hypotheses about query language properties. Although we observed all subactivities that we defined in the model, for the purpose of this experiment we could simplify the model of the query solving process according to Figure 4.10, “Simplified Model”

Figure 4.10. Simplified Model



This simplified model is still very similar to the top level diagram of our initial model. See also Figure 3.1, “Query Solving Process: Top Level”. The following changes were made. First, the test expression activity is merged with the correction effort because these activities were very similar and too difficult to distinguish. Second, within the construct deep structure activity, we distinguish directly between the understanding conditions of search and the understanding document structure activity. These are the only more detailed subactivities from the initial model that were required to answer the research questions in this experiment. Third, the correction efforts related to lexical problems, incorrect expressions, and clarify the result are merged to one correction effort type. With this model, we can model the behavior of querying subjects in this experiment and answer the research questions. The initial model is not necessarily incorrect, but it was too detailed for our purposes.

4.6. Conclusions

In this experiment we explored the model of the query solving process that was formulated in the previous chapter with a thinking aloud experiment. We found strong indications for performance differences between XQuery, XSLT and SQL/XML on complex queries. Performance was highest with XQuery, intermediate with XSLT and lowest with SQL/XML. The performance on simple queries was higher than on difficult queries and, more interestingly, the performance difference between the query languages on simple queries was negligible. This suggests that performance differences between XML query languages become more apparent with complex queries. The satisfaction of XQuery and XSLT was similar, and the satisfaction of SQL/XML was lowest. We also were able to explain the behavior of the subjects in terms of the query solving model. In addition, we found that the number of activities in the model, or the efficiency of the query solving process, is related to the correctness of the queries.

We found indications of the following XML query language properties that are likely to influence the user performance during the query process: consistency, verboseness, expression type, and Xpath embedding. We must note that this experiment consisted of a small number of subjects, so the differences between XSLT, XQuery and SQL/XML need to be validated with a larger group of subjects. The main explanation for the differences in performance between XSLT and XQuery, and SQL/XML is the lack of consistency in the latter. In fact, performance with SQL/XML and general satisfaction of this language was so low that we will not include this language in the remainder of this study. Further experiments are required to examine if the factors verboseness, Xpath embedding, and expression type are relevant factors that influence the performance with XML query languages. The level of procedurality may be another factor that influences the performance with a language, but we found no support for this hypothesis in this experiment. Therefore, procedurality will not be considered in the remainder of this study. However, it is possible that the non-procedural nature of XSLT remained implicit because the subjects did not use multiple template rules.

In the next chapter we try to validate these findings with an experiment with a larger number of subjects.

Chapter 5. Experiment 2: Validation

5.1. Introduction

The goal of the experiment described in this chapter is to validate the indications that we found in the thinking aloud experiment. We want to answer the following questions:

1. Is the usability, i.e. the user performance and satisfaction, of XQuery higher than XSLT?
2. Are performance differences between XQuery and XSLT larger with more complex queries compared to less complex queries?
3. Are Xpath embedding, verboseness, and the use of compound vs. specialized expressions relevant predictors for query performance?

We will try to answer the first and second question by comparing the subject's performance with XSLT and XQuery on a representative set of query tasks that is similar to the set of query tasks from the previous experiment, called *natural query tasks*. To assess the satisfaction of XSLT and XQuery, we ask the subjects to answer a questionnaire. For the third question, we compare the performance of subjects with XQuery and XSLT on the natural query tasks and with the subject's performance on a complementary set of query tasks, called the *directed query tasks*.

In our previous experiment we found the following indications of XML query language properties that are likely to influence the user performance during the query process:

1. Consistency: we noticed that the performance with an inconsistent language (SQL/XML) was generally lower compared to performance with more consistent languages (XSLT and XQuery). In addition, the subject's satisfaction of SQL/XML was very low. For these reasons, we exclude this language in the remainder of this study. XSLT and XQuery are equally consistent. Therefore, consistency cannot be used to distinguish between these languages.
2. Verboseness: we noticed that performance with the more verbose language was generally lower than the performance with one of the more compact languages. The more verbose language (XSLT) produces longer query expressions than the more compact language (XQuery).
3. Xpath embedding: the languages we compared in the previous experiment differed in the method of Xpath embedding. In the previous experiment, we noticed that the number of Xpath errors with XSLT was higher compared to XQuery. The difference is that XSLT mainly relies on context node evaluation, while XQuery relies on explicit variable bindings to embed Xpath expressions. We assume that

the latter is a more explicit method to bind Xpath expressions and that this is easier for users to understand.

4. Expression type: we noticed that performance on a specific task was higher when a specialized expression was available to solve the task. When a specialized expression was not available, the user had to use a combination of expressions to solve the problem. We noticed that the use of a specialized grouping expression in XSLT did led to higher user performance when solving a grouping problem compared to the use of compound expressions in XQuery.
5. Level of complexity of the question: we noticed that on less complex queries user performance differences were small between XSLT and XQuery. We also noticed that user performance differences became larger with more complex queries.

The user's background and experience are other factors that are likely to influence performance with an XML query language, but these were not varied in the previous experiment. In this experiment we will try to validate the results of our previous experiment with a larger group of subjects. We have the following research questions:

1. What is the difference in user performance on a representative set of tasks between XSLT and XQuery?
2. Which XML query language has the highest user satisfaction, XSLT or XQuery?
3. What is the influence of the complexity of the query task?
4. Is user performance lower with a more verbose language (XSLT) than with a more compact language (XQuery)?
5. Does the method of Xpath embedding with variables in XQuery lead to a better user performance compared with XSLT's method of evaluating context nodes?
6. Are compound expressions for a particular task easier to use than compound expressions? For instance, is the XSLT for-each-group easier to use than the compound expressions used in XQuery?

Based on these questions we formulate hypotheses that we will test in this experiment. We discuss the hypotheses and provide a short overview of the method of testing the hypotheses in the following sections.

5.1.1. Hypothesis 1: General User Performance

Based on the previous experiment, we expect that users perform better with XQuery than with XSLT:

- Hypothesis 1: *User performance on a representative set of queries is higher with XQuery than with XSLT.*

To answer the hypotheses in this experiment, we defined user performance in terms of effectiveness and efficiency. See also Chapter 2, *Background* for an overview of these terms. Effectiveness is the accuracy and completeness through which users achieve certain results. In this experiment, we established the correctness of the subject's result queries as a measurement for the effectiveness. Efficiency are the resources utilized in relation to accuracy and completeness through which users achieve certain results. In the previous experiment, we used the number of activities and transitions between activities during the query solving process as a measurement for efficiency. We also noticed that these numbers where highly correlated with the time spent on the query solving process. In this experiment, efficiency is represented by the time that subjects need to solve a task correctly, or at least partial correctly. We will discuss the performance measurements in more detail in Section 5.3, “Results”.

We use a similar set of five representative tasks as in the previous experiment: three selection queries with one or more conditions, a join query and a grouping query. The differences include the following. First, in the previous experiment we noticed that the highly experienced subjects encountered some problems with the XML document and DTD we used. These problems were mainly related to some peculiarities in the DTD. To avoid similar problems we used a very simple DTD and XML document in this experiment. Second, in the previous experiment we used three very similar simple queries. To answer hypotheses 4 - 6, we created some differences between the simple query tasks. The queries used in this experiment are discussed in more detail in Section 5.2.2.3, “Query tasks - Natural Condition”

5.1.2. Hypothesis 2: User Satisfaction

Based on the comments of subjects and the performance differences we noticed in the previous experiment, we expect that users will be more satisfied with XQuery compared to XSLT. The reason for this, is that we assume that higher performance will result in higher satisfaction. In addition, we expect that Xpath satisfaction will be higher for XQuery users compared to XSLT. In this experiment we will test whether these expectations are correct. The second hypothesis is:

- Hypothesis 2: *User satisfaction is higher with XQuery than with XSLT. Also, user satisfaction of Xpath is higher with XQuery than with XSLT.*

We used a questionnaire to gather information about the subject's satisfaction.

5.1.3. Hypothesis 3: Query Complexity

Earlier research indicates that performance differences between query languages become larger when the query complexity increases. We found similar indications in our previous experiment. In this experiment we want to validate these indications. The third hypothesis is:

- Hypothesis 3: *User performance differences between query languages will become larger on more complex queries compared to less complex queries.*

We used the same distinction between queries as made by Reisner [Rei77], Welty and Stemple [WS81], and Schlager[Schl91], namely between less complex and more complex queries. Less complex queries are selection queries with one or more conditions. More complex queries are, for instance, join and grouping queries. To test this hypothesis, we include the performance of the subjects on the natural query tasks.

5.1.4. Hypothesis 4: Verboseness

In Chapter 2, *Background* we discussed a common argument in the XML community against XSLT, namely the verbose syntax of this language. In the XML community it is argued that XSLT is difficult to learn and use among others for this particular reason. In the previous experiment we noticed that performance with XSLT indeed seemed to be lower compared to XQuery. Therefore, we expect that one reason why user performance is higher with XQuery compared to XSLT is that XSLT is more verbose compared to XQuery.

Verboseness is only one of the aspects in which XSLT and XQuery differ. In addition to verboseness, we expect that expression type and Xpath embedding are language aspects that cause user performance differences between XML query languages. The fourth hypothesis is as follows:

- Hypothesis 4: *User performance differences between XQuery and XSLT become smaller when verboseness is the only relevant difference between XQuery and XSLT expressions, but there will still be a difference.*

To test whether verboseness is an aspect that causes performance differences, we used one query task with two conditions:

1. *Natural condition*: for this query task subjects receive a description of the data that needs to be retrieved. For instance, "Find all names". In the previous experiment we used query tasks of this type exclusively. The correct XSLT and XQuery solutions of this task may differ in verboseness, Xpath embedding, and expression type. Subjects are completely free how they formulate a correct solution. This query is discussed in more detail in Section 5.2.2.3, “Query tasks - Natural Condition”.
2. *Directed condition*: this task describes the same query problem as the natural query task. However, for this type of task the subjects are directed towards a solution that emphasizes only one particular query language aspect, namely: verboseness and not in expression type and Xpath embedding. This is done by restricting the expressions that the subjects may use in the query task descriptions. How this is achieved is discussed in Section 5.2.2.4, “Query 1 - Directed Condition”.

We expect that the performance with XQuery will be better for both the natural and the directed query tasks. The performance with XSLT in the directed condition will increase more compared to XQuery because the expected negative effects of other aspects of XSLT are minimized. Therefore, we expect an interaction effect for language and direction where the performance difference between XQuery and XSLT becomes smaller in the directed condition because the difference here is only caused by the differences in verboseness between XSLT and XQuery.

5.1.5. Hypothesis 5: Xpath Embedding

In our previous experiment we noticed that more Xpath errors were made with XSLT compared to XQuery. We assume that the method of Xpath embedding in XQuery is easier for users to understand than the method used in XSLT. In XQuery, all Xpath expressions are bound to explicit named variables and the result of an Xpath expression can be checked directly by returning the value of the appropriate variable. In XSLT, Xpath expressions are embedded in XSLT elements and the result of an Xpath expression is dependent on the context node evaluation and can be checked less easily. In this experiment we will validate whether the difference in Xpath embedding between XSLT and XQuery leads to performance differences.

The fifth hypothesis is:

- Hypothesis 5: *User performance differences between XQuery and XSLT become smaller when Xpath embedding is the only relevant difference between XQuery and XSLT expressions, but there will still be a difference.*

We tested this hypothesis with two query tasks and we created for each task two conditions:

1. *Natural condition*: in this condition, the correct XSLT and XQuery solutions may differ in several aspects, namely verboseness, Xpath embedding, and expression type. Subjects are completely free how they formulate a correct solution. This query is discussed in more detail in Section 5.2.2.3, “Query tasks - Natural Condition”.
2. *Directed condition*: in this condition, the correct XSLT and XQuery solutions differ only in one aspect, namely Xpath embedding, and not in verboseness or expression type. Subjects are directed towards these specific correct solutions with some additional restrictions in the query task descriptions and by providing them with a partially solved query which we call a *template* of the query. See Section 5.2.2.5, “Query 2 - Directed Condition” and Section 5.2.2.7, “Query 4 - Directed Condition” for a complete description.

We use two query tasks to test whether Xpath embedding in XQuery leads to higher performance compared to XSLT: a selection query with multiple criteria and a query that includes a join problem..

For both the selection and the join query task in the natural condition, we expect that the performance differences between XQuery and XSLT in the natural conditions are due to verboseness, Xpath embedding, and possibly, the expression types that are used. Therefore, we expect for both query tasks that performance with XQuery will be better compared to performance with XSLT. In the directed conditions of these tasks, we expect to find only performance differences that are caused by Xpath embedding. The performance with XSLT in the directed condition will increase more compared to XQuery because the expected negative effects of other aspects of XSLT are minimized. Therefore, we also expect an interaction effect for language and direction where the performance difference between XQuery and XSLT becomes smaller in the directed condition.

5.1.6. Hypothesis 6: Expression Type

In the previous experiment we noticed that performance on a grouping query was higher for XSLT compared to XQuery. We assumed that the reason for this was the specialized `for-each-group` element in XSLT and the lack of an equivalent operator in XQuery. Therefore, we expect that the use of a specialized operator to solve a specific problem results in higher user performance. In this experiment we try to validate this expectation with two specialized expressions in XSLT and XQuery. We formulated two subhypotheses 6:

- Hypothesis 6a: *User performance on a specific query is higher with XQuery than with XSLT when a specialized expression for the problem is only available for XQuery.*
- Hypothesis 6b: *User performance on a specific query is higher with XSLT than with XQuery when a specialized expression for the problem is only available for XSLT.*

We test these hypothesis with two queries: a query that selects unique nodes and a query that includes a grouping problem. Both query tasks are described in Section 5.2.2.3, “Query tasks - Natural Condition”.

First, we discuss how hypothesis 6a is tested. We created for both XSLT and XQuery two conditions for this task:

1. *Natural condition*: in this condition, the correct XSLT and XQuery solutions may differ in several aspects, namely verboseness, Xpath embedding, and expression type. Subjects are completely free how they formulate a correct solution. For this particular query, a specialized expression to select unique nodes is available for both XQuery and XSLT. In addition, XSLT provides a solution that uses compound expressions.
2. *Directed condition*: in this condition, the correct XSLT and XQuery solutions differ only in one aspect, namely expression type, and not in verboseness or

Xpath embedding. A specialized expression is available for XQuery, but the use of this expression is prohibited for XSLT. Subjects are directed towards a specific correct solutions with some additional restrictions in the query task descriptions and with a template of the query. See Section 5.2.2.6, "Query 3 - Directed Condition" for a complete description.

For this query task, we expect better performance with XQuery compared to XSLT on both the directed and the natural task. We also expect that the directed condition will have a positive effect on performance with XQuery and a negative effect on performance with XSLT. We assume that in the directed condition performance with XSLT will be lower compared to the natural condition because subjects in the natural condition are likely to use a specialized expression and this is prohibited in the directed condition.

Second, hypothesis 6b is tested in a similar way. We created for both XSLT and XQuery two conditions for this task:

1. *Natural condition*: in this condition, the correct XSLT and XQuery solutions may differ in several aspects, namely verboseness, Xpath embedding, and expression type. Subjects are completely free how they formulate a correct solution. For this particular query, a specialized grouping expression is available only for XSLT. XQuery provides a solution for the grouping problem that uses compound expressions.
2. *Directed condition*: in this condition, the correct XSLT and XQuery solutions differ only in one aspect, namely expression type, and not in verboseness or Xpath embedding. A specialized grouping expression is available for XSLT, but not for XQuery. Subjects are directed towards a specific correct solution where only expression type is a relevant difference with additional restrictions in the query task descriptions and with a template of the query. See Section 5.2.2.8, "Query 5 - Directed Condition" for a complete description.

For this query task, we expect that performance with XSLT on both the natural and the directed query task is better compared to XQuery due to the specialized XSLT expression that can be used. We also expect that the direction has a more positive effect on performance with XSLT compared to performance with XQuery. In the directed condition, the only difference between the query language expressions is expression type. The influence of the other aspects, verboseness and Xpath embedding, are neutralized in both languages. We assume that this has a larger positive influence on the performance with XSLT than on the performance with XQuery.

5.2. Method

5.2.1. Subjects

The subjects were 74 students Information Science that participated in an introduction course on XML and related standards called "Exchange Languages". This course is part of the first year's curriculum of the study Information Sciences at Utrecht University. The course deals with the basics of XML, applications and background of XML, and standards like Xpath, XSLT, XQuery, XML Schema. The total duration of the course is 12 weeks and students were expected to spend approximately 20 hours per week on this course.

This experiment was embedded within the course as an additional practical assignment. Participation is voluntary and most of the participating students cooperated in the experiment. Students that participated are rewarded with an additional mark for their practical assignments, so they could receive a higher score for the course if they participated. Differences between the subject's background are determined with a questionnaire.

5.2.2. Materials

5.2.2.1. Query Languages

In this experiment, the languages XQuery 1.0 and XSLT 2.0 as implemented by Saxon 7.8 [Saxon] were used.

5.2.2.2. Questionnaire

We used two questionnaires in this experiment. First, a questionnaire to determine the background of the subjects. With background we mean for instance prior user experience with related languages and techniques, and preparation time of the subjects for the experiment. This questionnaire consisted of both binary questions and 1 - 7 point scale questions and is listed in Appendix B, *Questionnaire - Prior Experience*. Second, a questionnaire to determine the satisfaction of subjects about the query languages they used during the experiment. This questionnaire consisted of 1 - 7 scale questions exclusively. Both questionnaires are listed in Appendix C, *Questionnaire - Satisfaction*.

5.2.2.3. Query tasks - Natural Condition

The user performance with XML query languages is measured relative to a set of query tasks. We expected that subjects were able to solve approximately three less

complex and two more complex queries in one hour and therefore we constructed a set of five representative queries for this experiment. For each query task we also created a directed variant that isolates a query language aspect. These are discussed in the following sections. The query tasks are related to the document presented in Appendix D, *Experiment 2: XML Document*. The natural query tasks are for both languages as follows:

- *Query 1:* write a query that selects the following: the "author" elements with an attribute "period" with the value "classical"
- *Query 2:* write a query that creates a result element that contains the elements "name" and "nationality" of the "author" element that has:
  - a "nationality" that has *not* the value 'Greek';
  - a "nationality" that has *not* the value 'English';
  - an attribute "born" with value '1885'.
- *Query 3:* create a list element of unique nationality elements. Every nationality can therefore occur only once.
- *Query 4:* provide the "name" elements of the authors of which the "born" attributes have the same value as the "died" attribute with the author with "name" value 'Victor Hugo'.
- *Query 5:* Group "author" elements by "nationality" and provide both "nationality" as well as "name" elements in the group.

Query 1-3 are less complex tasks and query 4-5 are more complex tasks. Below we provide an overview of which query task is used to test which hypothesis:

Table 5.1. Hypotheses and Query Tasks

Hypothesis	Performance on Query	Natural or Directed
Hypothesis 1	1-5	Natural
Hypothesis 2	none (questionnaire)	none (questionnaire)
Hypothesis 3	1-3 and 4-5	Natural
Hypothesis 4	1	Natural vs. Directed
Hypothesis 5	2 and 4	Natural vs. Directed
Hypothesis 6	3 and 5	Natural vs. Directed

In the following sections we discuss the directed query tasks.

5.2.2.4. Query 1 - Directed Condition

Query 1 is a selection query with two criteria that is used to test hypothesis 4:

- *User performance differences between XQuery and XSLT become smaller when verboseness is the only relevant difference between XQuery and XSLT expressions, but there will still be a difference.*

The aspect of verboseness is isolated as follows. In the directed version of this query task, we tried to limit the number of Xpath expressions by restricting the number of XSLT elements in XSLT and prohibit the use of FLOWR expressions in XQuery. Neither XSLT or XQuery provides specialized operators for this task, so this aspect does not apply here. The directed query task descriptions for XSLT and XQuery are as follows:

- For XSLT: write a query that selects the following: the "author" elements with an attribute ""period" with the value "classical". Use <xsl:template match="text()"/> to override the default template rule that copies all text() nodes to the output. Use maximally one additional template rule.
- For XQuery: write a query that selects the following: the "author" elements with an attribute ""period" with the value "classical". Do not use FLOWR expressions.

A correct solution for XSLT is:

```
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="2.0">
<xsl:output indent="yes"/>
<xsl:template match="text()" />

<xsl:template match="/">
 <xsl:copy-of select="//author[@period='classical']" />
</xsl:template>
</xsl:stylesheet>
```

A correct solution for XQuery is:

```
//author[@period='classical']
```

As we can see, the main difference between the XSLT and the XQuery solution is the verboseness of the solution. The XQuery solution is one basic Xpath expression while the XSLT solution is one Xpath expression embedded in a minimal XSLT sheet.

To test hypothesis 4, we compare the subject's performance on the directed condition of query 1 with the performance on the natural condition of query 1. We expect a main effect for language, where the user performance is better with XQuery than with XSLT on both the natural and the directed condition. Also, we expect an interaction effect

between language and direction where the difference in user performance is lower in the directed condition. The reason for this is that the direction will have a more positive effect on XSLT than on XQuery because of the neutralization of Xpath embedding.

5.2.2.5. Query 2 - Directed Condition

Query 2 includes multiple selection criteria and is used, together with query 4, to test hypothesis 5:

- *User performance differences between XQuery and XSLT become smaller when Xpath embedding is the only relevant difference between XQuery and XSLT expressions, but there will still be a difference.*

The multiple selection criteria makes this query suitable to isolate Xpath embedding from verboseness and expression type. The only intended relevant difference between the XQuery and the XSLT solution for query 2, is that the Xpath expressions are embedded in a different way in these languages.

The factor Xpath embedding is isolated in this query task as follows. First, the tasks description of query 2 includes multiple selection criteria, and the expected query solutions will therefore consist of multiple Xpath expressions. We expect that the performance differences between XQuery and XSLT become larger with queries that need more Xpath expressions. In addition, in the directed condition of query 2, we prohibited the use of Xpath predicates. These predicates overlap with some constructs in both XSLT and XQuery and reduce the number of Xpath expressions that are necessary in the query solution. For instance, when a subject uses an Xpath predicate, there might be no more need to use a `WHERE` clause in XQuery or and `xsl:if` element in XSLT. We are specifically interested in these XSLT and XQuery constructs that embed Xpath expressions and therefore we prohibited the use of Xpath predicates. Second, in the directed query task we tried to exclude the influence of verboseness differences between the languages by providing a template of the query so that subjects do not need to formulate a complete query. The templates we provided are listed below. Note that it is possible that the verboseness of the XSLT template will make it more difficult to understand the XSLT template compared to the XQuery template and thereby has a negative influence on the performance with XSLT. However, we assume that users who are familiar with XSLT will have little problems with understanding the template. Third, neither the XSLT template, nor the XQuery template uses specialized operators for this task. Therefore, the difference between specialized vs. compound expressions does not apply here. The query task description for XSLT and XQuery in the directed condition is as follows:

- Complete the given template query *without* using Xpath predicates. Create a result element that contains the elements "name" and "nationality" of the "author" element that has:
  - a "nationality" that has *not* the value 'Greek';

  - a "nationality" that has *not* the value 'English';
  - an attribute "born" with value '1885'.
- Template for XSLT:

```
<xsl:transform
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="2.0">
<xsl:output method="xml" indent="yes"/>
<xsl:template match="text()" />

<xsl:template match="...">
<xsl:if test="...">
<result>
<xsl:copy-of select="..." />
...
</result>
</xsl:if>
</xsl:template>
</xsl:transform>
```

- Template for XQuery:

```
for ... in ...
where ...
return
<result>
{
... , ...
}
</result>
```

A correct solution for XSLT is:

```
<xsl:transform
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="2.0">
<xsl:output method="xml" indent="yes"/>
<xsl:template match="text()" />

<xsl:template match="author">
<xsl:if test="nationality!='Greek'
 and nationality!='English'
 and @born='1885'">
<result>
 <xsl:copy-of select="name" />
 <xsl:copy-of select="nationality" />
</result>
</xsl:if>
</xsl:template>
</xsl:transform>
```

A correct solution for XQuery:

```
for $a in //author
where $a/@born=1885
and $a/nationality!="Greek"
and $a/nationality != "English"
return
<result>
{
$a/name, $a/nationality
}
</result>
```

The most important feature of these solutions, is that multiple Xpath expressions are used. The influence of XSLT's verboseness is reduced by providing the subjects with a template of the query solution. Therefore, performance differences on the directed condition of this query task, are caused by differences in Xpath embedding between XQuery and XSLT.

We expect the following performance differences on query 2. First, we expect a main effect of language where user performance will be better with XQuery in both the natural and the directed condition. Second, we expect an interaction effect between language and direction. In the directed condition, performance with both XSLT and XQuery will increase compared with the natural queries because of the query template that is provided. However, this template will have a more positive effect for XSLT than for XQuery because the disadvantage of XSLT's verboseness is resolved.

5.2.2.6. Query 3 - Directed Condition

Query 3 selects unique nodes from the XML document and is used to test hypothesis 6a:

- *User performance on a specific query is higher with XQuery than with XSLT when a specialized expression for the problem is only available for XQuery.*

In query 3, subjects need to create a list with unique values from the XML document. During the course, student's learned to solve this type of problem in XSLT with a compound Xpath function based on a method described by Kay [Kay98]. In XQuery, they learned to use a specialized Xpath function for this task. The students were aware that they could also use this specialized Xpath operator in XSLT, therefore we had to prohibit the use of this operator in the directed condition of the XSLT query task. To compensate for XSLT's verboseness, we provided the subjects with a query template that consisted of a standard XSLT sheet. In addition, this XSLT sheet minimized the use of Xpath expressions. We also provided a small query template for the XQuery users to compensate for the advantage of subjects who received a XSLT template. The query task descriptions for both languages are as follows:

- For XSLT: Create a list element of unique nationality elements. Every nationality can therefore occur only once. Use Kay's solution to remove duplicate values! The following template is provided:

```
<xsl:transform
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 version="1.0">
<xsl:output method="xml" indent="yes"/>
<xsl:template match="/">
<list>
...
</list>
</xsl:template>
</xsl:transform>
```

- For XQuery: Create a list element of unique nationality elements. Every nationality can therefore occur only once. Use the Xpath function distinct-values(). The following template is provided:

```
<list>
{
...
}
</list>
```

Examples of correct solutions are as follows:

- XSLT:

```
<xsl:transform
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   version="1.0">
<xsl:output method="xml" indent="yes"/>
<xsl:template match="/">
<list>
<xsl:for-each select="//nationality[not(.=preceding::nationality)]">
<nationality>
 <xsl:value-of select="."/>
</nationality>
</xsl:for-each>
</list>
</xsl:template>
</xsl:transform>
```

- XQuery:

```
<list>
{
for $a in distinct-values(//nationality)
return
<nationality>{$a}</nationality>
}
</list>
```

The most important difference between the XSLT solution and the XQuery solution is the use of a compound Xpath expression in XSLT (`//nationality[not(.=preceding::nationality)]`) and the use of a specialized Xpath expression in XQuery (`distinct-values(//nationality)`). The influence of verboseness and of Xpath embedding is neutralized by the provided query templates.

We expect the following performance differences on this query. First, we expect a main effect of language, where performance with XQuery is better than with XSLT. We also expect that user performance with XQuery will increase in the directed condition, because subjects do not have to think about which expression must be used and a small template of the query is given. Second, we expect that user performance with XSLT in the natural condition is better than in the directed condition. In the natural condition we expect that subjects will use the same specialized Xpath expression that is used with XQuery. The positive effect of the query template in the directed condition will probably have less effect than the use of this specialized operator. Therefore, we expect an interaction effect for language and direction where the direction has a positive effect on performance with XQuery, and a negative effect on the performance with XSLT.

5.2.2.7. Query 4 - Directed Condition

Query 4 is a join query and is used, together with query 2, to test hypothesis 5:

- *User performance differences between XQuery and XSLT become smaller when Xpath embedding is the only relevant difference between XQuery and XSLT expressions, but there will still be a difference.*

The difference between query 2 and query 4 is the level of complexity of these query tasks. Query 2 is a selection query with multiple selection criteria while query 4 is a join query. Both queries consist of a larger number of Xpath expressions compared to the other queries. According to the distinctions made in earlier research, query 4 is more complex than query 2.

The factor Xpath embedding is isolated in this query task in a similar way as in query 2. First, the number of required Xpath expressions necessary to solve the query tasks is rather large. Second, in the directed condition subjects are not allowed to use Xpath predicates because these may reduce the number of Xpath expressions and overlap with XSLT and XQuery functionality. Third, in the directed conditions we provide templates of the query to reduce the differences in verboseness between the languages. The following task description and templates are used in the directed condition:

- Complete the following template by replacing the dots with the appropriate Xpath expressions. Do not use Xpath predicates! The query is as follows: Provide the "name" elements of the authors of which the "born" attributes have the same value as the "died" attribute with the author with "name" value 'Victor Hugo'.
- XSLT template:

```
<xsl:transform
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<xsl:output method="xml" indent="yes"/>

<xsl:template match="/">
<xsl:for-each select="...">
 <xsl:if test="...">
 <xsl:variable name="..." select="..."/>
 <xsl:for-each select="...">
  <xsl:if test="...">
  <xsl:copy-of select="..."/>
  </xsl:if>
 </xsl:for-each>
 </xsl:if>
</xsl:for-each>
</xsl:template>
</xsl:transform>
```

- XQuery template:

```
for ... in ...
where ...
return
 for ...
 where ...
 return ...
```

A correct solution for this query task in XSLT is:

```
<xsl:transform
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<xsl:output method="xml" indent="yes"/>

<xsl:template match="/">
<xsl:for-each select="//author">
 <xsl:if test="name='Victor Hugo'">
 <xsl:variable name="vh" select="."/>
 <xsl:for-each select="//author">
  <xsl:if test="@born=$vh/@died">
  <xsl:copy-of select="name"/>
  </xsl:if>
 </xsl:for-each>
 </xsl:if>
</xsl:for-each>
</xsl:template>

</xsl:transform>
```

A correct solution for this query task in XQuery is:

```
for $vh in //author
where $vh/name="Victor Hugo"
return
 for $a in //author
 where $a/@born = $vh/@died
 return $a/name
```

The most important feature of these solutions, is that multiple Xpath expressions are used. The influence of XSLT's verboseness is reduced by providing the subjects with a template of the query solution. Therefore, performance differences on the directed condition of this query task, are caused by differences in Xpath embedding between XQuery and XSLT. Note that the XSLT solution we propose includes one explicit variable binding to include an Xpath expression. This is very similar to the variable bindings used in XQuery. We would have preferred to exclude explicit variables in XSLT completely, but we cannot think of an alternative solution to solve a join query without using variables in XSLT. We expect that the influence of one explicit variable binding in an XSLT solution will hardly influence the overall performance on this query task.

We have the same expectations regarding performance as with query 2. First, user performance will be better with XQuery in both the natural and the directed condition. In the directed condition, performance with both XSLT and XQuery will increase compared with the natural queries because of the query template that is provided.

However, this template will have a more positive effect for XSLT than for XQuery because the disadvantage of XSLT's verboseness is resolved.

5.2.2.8. Query 5 - Directed Condition

Query 5 is a grouping query and is used to test hypothesis 6b:

- Hypothesis 6b: *User performance on a specific query is higher with XSLT than with XQuery when a specialized expression for the problem is only available for XSLT.*

Hypothesis 6b is strongly related to hypothesis 6a, that is tested with query 3. The difference between query 3 and query 5 is twofold. First, the queries differ in level of complexity. Query 3 is a query that selects unique nodes and is not very complex. Query 5 is a grouping query and is more complex. Second, in query 3 the specialized operator is used with XQuery and in this query the specialized operator is used with XSLT.

XQuery has no specialized grouping operator and grouping queries need to be solved with either a combined for and let clause or with two nested for clauses. XSLT 2.0, on the other hand, introduced the element `for-each-group` to solve grouping problems because the XSLT 1.0 solution with compound expressions is generally perceived as very difficult to use. In the directed condition we tried to equalize the query language expression with respect to the other factors. To compensate for XSLT's verboseness we provided a templates of the queries. We expected that the difficulties with Xpath embedding in XSLT would be of little influence because the number of Xpath expressions is relatively small for this query. The query tasks in the directed conditions are as follows:

- For XSLT: complete the following template and use the `for-each-group` element to solve this query. The query is: Group "author" elements by "nationality" and provide both "nationality" as well as "name" elements in the group.

```
<xsl:transform
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="2.0">
<xsl:output method="xml" indent="yes"/>
<xsl:template match="/">
<result>
...
</result>
</xsl:template>
</xsl:transform>
```

- For XQuery: complete the following template and use either a for and let clause or a combination of for clauses to solve this query. The query is: Group "author" elements by "nationality" and provide both "nationality" as well as "name" elements in the group.

```
<result>
{
...
}
</result>
```

A correct solution for this query task in XSLT is:

```
<xsl:transform
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="2.0">
<xsl:output method="xml" indent="yes"/>

<xsl:template match="/">
<result>
<xsl:for-each-group select="//author" group-by="nationality">
<nationality_list>
 <nationality>
 <xsl:copy-of select="current-grouping-key()"/>
 </nationality>
 <xsl:copy-of select="current-group()/name"/>
</nationality_list>
</xsl:for-each-group>
</result>
</xsl:template>
</xsl:transform>
```

A correct solution for this query task in XQuery is:

```
<result>
{
for $n in distinct-values(//nationality)
let $p := //author[nationality=$n]
return
<nationality_list>
 <nationality>{$n}</nationality>
 {$p/name}
</nationality_list>
}
</result>
```

The most important difference between the XSLT solution and the XQuery solution is the use of a specialized grouping operator in XSLT, i.e. `<xsl:for-each-group>`, and the use of a several compound expressions in XQuery, i.e. `for`, `let`, `distinct-values()`. The influence of verboseness is neutralized by the provided query templates and the effect of Xpath embedding is probably small because of the relatively low number of Xpath expressions.

We expect the following performance differences between XSLT and XQuery and between the natural and directed conditions. First, we expect a main effect of language. Overall performance on both conditions will be better with XSLT compared to XQuery because of the specialized grouping expression in XSLT. Second, we expect an interaction effect between language and direction. The influence of the direction will have a larger positive effect on the performance with XSLT compared to XQuery because the disadvantage of XSLT's verboseness is resolved with the query template.

5.2.2.9. Experimental Platform

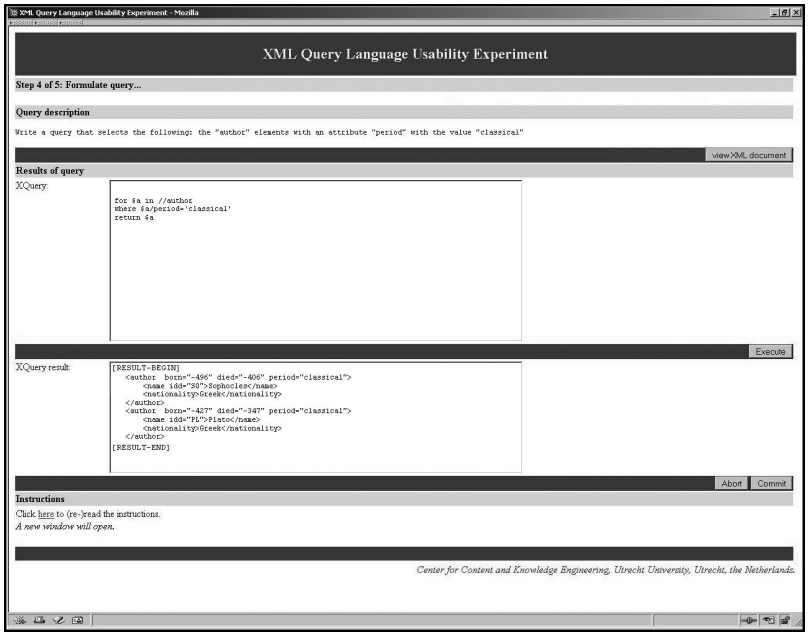
We use the XML Testbed in this experiment. This experimental platform is an instance of the Testbed for the Evaluation of Retrieval Systems (TERS) [VZw04]. TERS provides a web interface to perform both usability as well as retrieval experiments. The advantage of the XML Testbed for subjects is that they can fill in questionnaires, practice with the query languages, formulate query tasks, view the XML source documents and the result queries within one interface. Both XSLT and XQuery processors can be embedded within the system and therefore the same interface can be used for both XSLT and XQuery. The testbed enables researchers to capture the answers to questionnaire questions as well as the result queries that subjects submit to the query processor together with the time intervals between each submission. The correctness of final result queries can be scored for correctness by the researcher. Therefore, both the effectiveness and the efficiency of subjects and the answers to the questionnaire questions can be determined easily and fast. Also, the results of the experiments can be reproduced by the experimenter or third parties because all results are stored in a postgres database. Another advantage of the XML Testbed is that

multiple subjects can simultaneously participate in one experimental session. The testbed consists of the following pages:

1. Welcome and registration page;
2. General questionnaire about the subject's experience;
3. Drop down list which contains the query tasks that were assigned to the registered subject;
4. Query interface that consists of the following:
    a. Query task description and the reference to the related XML document.
    b. A button which opens the related document when pressed on;
    c. A text field in which the subject is supposed to enter the correct query statement, either with XQuery of with XSLT;
    d. A button "execute" that submits the query statement to the query processor.
    e. A text field in which the result of the query or the error messages appears after pressing the button "execute".
    f. A button "abort" that the subject is supposed to use if he wants to pause or when he wants to try another query. After pressing "abort", the subject returns to the drop down list with queries. The aborted query still remains in the list and when the subject chose this query again, the aborted query and the last result is shown again.
    g. A button "commit" that the subject was supposed to use if he thought that the query was correct. After pressing "commit", the subject returns to the drop down list with the query tasks and the committed query task is removed from the list.
5. Questionnaire about the satisfaction of the subject about the query languages (Xpath and XQuery or XSLT) he has used.
6. A "Thanks for the participation" page.

Here is a screen shot of the Query interface:

Figure 5.1. Screenshot of the Query Interface



5.2.3. Procedure

In this experiment we compared the user performance with XSLT and XQuery. For practical reasons and to avoid confusion between the languages we decided to let the subjects work with only one query language. To test the hypotheses about the query language aspects, two types of query tasks, natural and directed, are required for both languages. Therefore, four groups of subjects are needed to test our hypotheses. Subjects may have an advantage with directed query tasks because additional information is provided about which expression they must use, sometimes together with template of the query. For this reason we provided both types of query tasks to each subject instead of providing a group with only directed or natural tasks. We made the following division of subject groups and query tasks:

Table 5.2. Division of Subjects and Tasks

		Less complex tasks			More complex tasks	
Group	**Language**	**Query 1**	**Query 2**	**Query 3**	**Query 4**	**Query 5**
1 (N=19)	XSLT	Natural	Natural	Directed	Directed	Natural
2 (N=19)	XSLT	Directed	Directed	Natural	Natural	Directed
3 (N=18)	XQuery	Directed	Directed	Natural	Natural	Directed
4 (N=18)	XQuery	Natural	Natural	Directed	Directed	Natural

As we can see in this table, subjects who were assigned to Group 1 work with XSLT, had three natural tasks (query 1, 2, and 5) and two directed tasks (query 3 and 4). Subjects were assigned to groups based on the alphabetic order of their first names. Each group consists of 18 or 19 subjects. When a subject logged in to the XML Testbed interface, only the tasks and the language that belongs to his or her group are presented.

The experiment is embedded in the curriculum of the course 'Exchange Languages'. Participants of the course attended lectures, practiced with XML and related standards, and studied literature. In the first week they learned about XML and DTDs. In the second and third week they learned Xpath and XSLT. In the fourth week they learned XQuery. The experiment was conducted in the fifth week. Note that practical assignments were compulsory for Xpath and XSLT but not for XQuery.

We organized six sessions with a total of 74 subjects. The sessions were conducted in the regular student's practice rooms of the Information & Computing Sciences group. Students worked on regular, modern PC's with Windows XP and MS Internet Explorer. The number of subjects per session varied from 6 to 20. Each session lasted three hours and were organized as follows:

1. 5 minutes: subjects were welcomed and received short general instructions about the experiment. The instructions contained in particular procedural information, e.g. subjects had to work alone, were not allowed to communicate with other students, were not allowed to use cut and paste functionality, and were only allowed to use reference materials from the course exclusively. We advised them to work for a maximum of 10 minutes on one query and continue to the following task if they do not succeed. The instructions also contain the URL of the XML Testbed interface, as well as their login and password.
2. 5 minutes: after the subjects logged in, they filled in the questionnaire to determine their relevant experience and their preparation time for the experiment. See also Appendix B, *Questionnaire - Prior Experience*.
3. 60 minutes: to compensate for their lack of practical experience with XQuery, the subjects practiced for one hour with this language. They were allowed to try out the examples that were discussed in the lecture and were presented in literature. Note that the XSLT groups also practiced with XQuery because we

wanted that all subjects had similar levels of experience with all languages. However, after 45 minutes the XSLT groups received a brief description of two XSLT 2.0 features that were not previously discussed in the course but were necessary for the experiment. They were allowed to practice with these features for 15 minutes.

4. 15 minutes: after the practice hour, the subjects started with the real query tasks. All groups started with the same set of five Xpath tasks to determine the baseline performance of each subject. They were allowed to solve these tasks for 15 minutes. These tasks are listed in Appendix E, *Baseline Performance Tasks*
5. 90 minutes: The rest of the session was mainly spent on either the five XSLT or XQuery tasks.
6. 5 minutes: The session was concluded with a questionnaire about the satisfaction of the languages used. See also Appendix C, *Questionnaire - Satisfaction*

5.2.4. Design and Analysis

In this section we provide an overview of the design, variables, scoring methods, and statistical analyses that are used to test the hypotheses in this experiment. We hypothesized that several factors cause performance and satisfaction differences. The factors that we expect to cause usability differences are the *independent variables* in this experiment. The *dependent variables* are the values that we try to predict: performance on certain query tasks and the subject's satisfaction of XSLT and XQuery.

XML query language is an independent variable in this experiment. This variable has two values: XQuery and XSLT. Because of the limited available time for the experiment, we decided to let subjects work with either XQuery or XSLT exclusively. When subjects are assigned to one value of a variable exclusively, this variable is called a *between-subjects* variable. In this experiment, query language is an independent, between-subjects variable. When subjects are assigned to multiple values of a variable, this variable is called a *within-subjects* variable. We also expected that the level of complexity of a query task would influence the performance with a query language. We distinguished between less complex query tasks and more complex query tasks and all subjects were supposed to solve both types of tasks. Complexity is therefore an independent, within-subjects variable.

We expect that three query language aspects are causing usability differences: verboseness, expression type, and Xpath embedding. To test the correctness of our expectations, we compare the performance differences between-subjects working with XSLT and XQuery on two conditions: natural query tasks and directed query tasks. As explained earlier, the natural query tasks consist of a query task description and the subjects are required to formulate an appropriate query language expression. In this condition, subjects are completely free how they formulate the query and they receive no additional instructions. In the directed condition of a query task, we neutralized all relevant differences between query language aspects except for one

with additional requirements. These requirements included restrictions about the expressions that could be used and with partially solved queries, called templates, that the subjects need to complete. For each of the query tasks 1 - 5, there is a natural condition and a directed condition. Each subject is assigned either to a natural or a directed condition of a query task. We name this variable *direction* and this is an independent, between-subjects variable.

The different aspects of usability are the *dependent variables* in this experiment. Usability is subdivided in performance (effectiveness and efficiency) and satisfaction. The values of these dependent variables are determined as follows. First, the effectiveness is the correctness of the subject's result queries. The experimenter rated all result queries after the experiment as follows:

- *Correct:* result queries were considered as correct when the correct result document was retrieved and the query expression was according to the query task description. This latter was required in particular for the directed query tasks. Correct queries were rated with *2*.
- *Essentially correct:* result queries were considered essentially correct when the most difficult part of the result query was correct, but a minor error was made or a small part of the query was forgotten. For instance, when the subject needed to retrieve a list of unique elements and the subject instead retrieved a list of unique values. Essentially correct queries were rated with *1*.
- *Incorrect or no attempt:* all other result queries or empty results were considered incorrect. These queries were rated with *0*.

Second, efficiency is represented by the time that the subject needed to create a correct or essentially correct solution of the query task. The time to complete a task was registered automatically. Third, satisfaction is the subjective experience of the user of the query language and is measured with a 1-7 point scale questionnaire. See also Appendix C, *Questionnaire - Satisfaction*.

We now provide a short overview of the statistical tests that were used to test the hypotheses in this experiment. The choice for a test depends on which type of variables are included in the hypothesis. Most of the tests are based on (or are a variant of) Analysis of Variance tests (Anova). An Anova is used to test if more than two mean values are significantly different. To compare two mean values a t-test can be used. For instance, to test whether the four groups of subjects we described in the previous section (see Table 5.2, “Division of Subjects and Tasks”) differ in prior experience we used Anova tests. More specifically, we used One-way, between subject Anova tests because we compared the mean values of one dependent, between-subjects variable. In addition to the Anova tests, we used one paired t-test to compare mean preparation time of XQuery with the mean preparation time of XSLT for all subjects. In this case, we used a t-test because we only needed to compare two mean values. All subjects spent time on learning XSLT and XQuery and therefore the preparation time is a within-subjects variable and a paired t-test is required.

We expect that users with more expertise with programming languages, database and XML related standards will perform better compared to users with less expertise. In this experiment we focus on query language aspects only and therefore we would like to exclude other possible influences on user performance. It is likely that subjects with more expertise and with more preparation time will perform better independently of which query language is used. Therefore, we would like to control for these factors by using a variant of the Anova, namely Analysis of Covariance (Ancova). In an Ancova the factors that need to be controlled are included as *covariates* and their influence on the dependent variable is removed from the analysis.

We used Ancova's to test hypotheses 1 - 6 and included expertise and preparation time of the subjects as covariates. For hypothesis 1 we tested whether the mean performance with XQuery was higher compared to performance with XSLT. In this analysis one between-subjects independent variable is included, namely language, together with two covariates: expertise and preparation time. The dependent variables are effectiveness and efficiency. The appropriate test is a one way, between-subjects Ancova and the results are presented in the following section. For hypothesis 2 we tested whether the satisfaction of XQuery was higher compared to XSLT and whether the satisfaction with Xpath was higher with XQuery users than with XSLT users. We used a similar one way, between-subjects Ancova as for hypothesis 1 only with the dependent variable satisfaction instead of performance. For hypothesis 3 we used a different design than with hypotheses 1 and 2. Hypothesis 3 included two variables instead of one, namely XML query language and task complexity. We tested for differences in mean performance in four conditions: XSLT and XQuery performance on both less complex and more complex query tasks. This means that two variables with two values are included in the analysis: language and complexity. We stated earlier that language is a between-subjects variable and complexity is a within-subjects variable. The test that includes these type of variables is a 2 x 2 Mixed Ancova test, where expertise and preparation time of the subjects are included as covariates. For hypotheses 4 - 6, we also tested for mean performance differences in four conditions. Namely, performance with XSLT and XQuery on both a natural task and a directed task. Both variables have two values and both are between subject variables. The appropriate test for these variables is a 2 x 2 between subject Ancova, again with expertise and preparation time as covariates.

The outcome of Anova and Ancova tests are presented as the *F-ratio* and a *p-value*. These are to be interpreted as follows. The F-ratio is a measurement for the variance that is due to the influence of independent variable divided by the error variance. For instance, the independent variable is the query language assigned to the subjects and the error variance is the variance in scores due to individual differences between subjects. When the F-ratio is smaller than 1, the variance caused by the influence of the independent variable is smaller than the variance that is due to error. Therefore, the result of direction is not significant when the F-ratio is smaller than 1. The larger the F-ratio, the larger the effect of the direction compared to the error variance in the data. The degrees of freedom are used to calculate the F-ratio and are given between

parentheses. The p-value is the probability of getting the F-ratio by chance alone. When the p-value is smaller than .05, the F-ratio is regarded as significant. The results of these analyses are presented in the following section.

5.3. Results

In this section we present the results of this experiment. We start with an overview of the results for the expertise, preparation time, and performance on the baseline Xpath tasks of the subjects. Than the results that are related to hypotheses 1 - 6 are presented.

5.3.1. Prior Experience of the Subjects

5.3.1.1. Expertise

One of the topics in the questionnaire was the subject's experience with XML, HTML, programming languages and number of related courses they followed, namely Imperative Programming and an introduction course to relational Databases. After the experiment we also checked at the faculty's office for the student's academic achievements on these courses. The questionnaire listed in Appendix B, *Questionnaire - Prior Experience* was used to determine the student's general experience. We used this information to create a measurement for the relevant expertise of the subjects. This measurement consists of the following items:

1. Experience: number of courses, success with these courses, personal experience with related languages;
2. Last achieved score for the course *Imperative Programming*;
3. Last achieved score for the course *Databases*.

So, the experience of subjects is measured as the mean value of thirteen 1 - 7 point scale items and the last achieved scores on the courses Imperative Programming and Database are scaled from 1 - 10. We combined these measurements to one general expertise measurement by taking the mean value of the three items mentioned above. Four one-way between-subjects Analysis of Variance (Anova) tests were used to check for differences in the mean values of these items between the four groups. The following results were found:

1. Experience: $F(3,70) = 1.02$, $p < .39$
2. Imperative Programming: $F(3,45) = .17$, $p < 1.00$
3. Databases: $F(3,67) = 1.23$ $p < .31$
4. Expertise: $F(3,70) = .71$, $p < .55$.

The results that are presented above show that no significant differences were found on any item. Therefore, the subjects in the four groups do not differ in experience

level as measured with the questionnaire, nor for the scores on the courses Imperative Programming and Databases, nor in the combined expertise measurement.

5.3.1.2. Preparation Time

Before the test, we asked participants to fill in a questionnaire. One of the topics of this questionnaire was the learning and practice time spent on the query languages in the course. See also Appendix B, *Questionnaire - Prior Experience*. We use the subject's answers to these questions to find the learning and practice time of Xpath, XSLT and XQuery. We counted the presence of a student at a lecture as one hour of preparation time. For the whole group we found the following mean preparation time:

1. Xpath: 4,9 hours
2. XSLT: 4,8 hours
3. XQuery: 2,9 hours

We combined the learning and practice time of the three languages to one general learning and practice measurement by taking the mean value of preparation time spent on Xpath, XSLT and XQuery.

We checked whether the four groups differed in learning time with four one-way between-subjects Anova tests. No significant differences were found between the four groups on any language, nor on the combined preparation time:

1. Xpath preparation: $F(3,70) = 1.83$, $p < .15$
2. XSLT preparation: $F(3,70) = .75$, $p < .53$
3. XQuery preparation: $F(3,70) = .10$, $p < .96$
4. Combined preparation: $F(3,70) = 1.12$, $p < .35$

Therefore, we conclude that there are no differences between the four groups regarding preparation time of Xpath, XSLT, and XQuery. We did notice that the difference in preparation time between XSLT and XQuery is quite large. We used a paired samples t-test to test whether the differences between preparation time for XQuery and XSLT were significant for the whole group of subjects. We found that the XSLT preparation is significantly longer than the XQuery preparation ($t= 11.60$, $df = 73$, $p < .001$, one-tailed).

5.3.1.3. Xpath Performance

All participants made five simple Xpath queries to test their baseline performance. The queries were judged as correct or incorrect and the time for each query per participant was measured. As a Xpath efficiency measurement we used the average time per query.

We checked whether the four groups differed in Xpath performance with three one-way Anova tests. No significant differences were found between the four groups for the performance differences:

- Mean correctness: $F(3,70) = 1.46$, $p < .23$
- Mean time: $F(3,70) = .37$, $p < .78$

Therefore, we conclude that there is no significant difference in performance of the different groups on the Xpath tasks.

In the following sections we present the results for hypotheses 1 - 6.

5.3.2. Hypothesis 1: General User Performance

In this section we provide the results for the first hypothesis:

- *User performance on a representative set of queries is higher with XQuery than with XSLT.*

For each subject we computed the mean correctness for all natural tasks (Query 1 - 5) to determine the performance differences between XSLT and XQuery on the representative set of tasks. The results of the directed tasks were excluded from the analysis because the directed tasks are created to answer hypothesis 4-6 and are not considered to be representative, natural query tasks. See the following table with the mean values, standard deviations (Stdev.), and number of subjects (N) for correctness for all natural queries per language:

Table 5.3. Mean Correctness

Language	**Mean**	**(Stdev.)**	**N**
XSLT	.81	(.58)	38
XQuery	1.09	(.52)	36

We expected that both expertise and preparation time of the user influences performance with a language. To control for the influence of user's expertise and preparation time, we included these factors as covariates in an Ancova test. This analysis shows us that the difference in mean correctness over the complete set of queries is significant ($F(1,70) = 9.02$, $p < .004$).

The other performance indicator that we considered is efficiency, or the time spent on query tasks that are solved correct or essentially correct. The mean time for the

natural tasks was computed for each subject. See the following table with the mean values for time spent on the natural tasks per language:

Table 5.4. Mean Time

Language	Mean	(Stdev.)	N
XSLT	657	(475)	32
XQuery	558	(343)	34

Again, we tested this difference with an Ancova test, and noticed that this difference is not significant ($F(1,62) = .64$, $p < .43$).

Subjects solved more queries correctly with XQuery than with XSLT in this experiment, but there were no differences with respect to query solving time between the languages. Based on these results, we accept hypothesis 1.

5.3.3. Hypothesis 2: User Satisfaction

Hypothesis 2 states the following:

- *User satisfaction is higher with XQuery than with XSLT. Also, user satisfaction of Xpath is higher with XQuery than with XSLT.*

We tested this hypothesis as follows. After the test, we asked subjects to complete the questionnaire listed in Appendix C, *Questionnaire - Satisfaction*. See the following table with the values for XSLT and XQuery. Note that three subjects forgot to fill in this questionnaire and therefore the number of subjects is 71 (N=71).

Table 5.5. Query Language Satisfaction

Language	Mean	(Stdev.)	N
XSLT	3.65	(1.02)	36
XQuery	4.07	(1.24)	35

We used the same covariates as in the previous analysis and this showed us that the difference between the satisfaction of XSLT and XQuery is nearly significant: $F(1,67) = 3.79$, $p < .09$

See also the following table with the mean values for Xpath satisfaction of XSLT and XQuery:

Table 5.6. Xpath Satisfaction

Language	Mean	(Stdev.)	N
XSLT	4.52	(.97)	36
XQuery	4.90	(.76)	35

An Ancova test showed that the difference between the satisfaction of Xpath for the XQuery and the XSLT group is nearly significant: $F(1,67) = 3.63$, $p < .06$

Based on these results we conclude that the satisfaction of XSLT is lower than satisfaction of XQuery. Also, the satisfaction of Xpath is lower for XSLT users than for XQuery users. We accept hypothesis 2.

5.3.4. Hypothesis 3: Query Complexity

In hypothesis 3 we stated the following:

- Hypothesis 3: *User performance differences between query languages will become larger on more complex queries compared to less complex queries.*

In this experiment the subjects tried to find correct solutions for five queries that consist of three less complex queries (query 1 - 3) and two more complex queries (query 4 -5). We computed for each subject the mean correctness and the mean efficiency on both types of tasks for both XSLT (N=38) and XQuery (N=36). We included only natural query tasks. The table below shows the estimated mean values and the standard error of the mean (Sem) for correctness. The estimated mean values are corrected for the influences of preparation time and expertise of the subjects:

Table 5.7. Correctness - Complexity

Language	Mean less complex (Sem)		Mean more complex (Sem)		Total (Sem)	
XSLT	1.12	(.110)	.46	(.122)	.79	(.083)
XQuery	1.43	(.113)	.63	(.125)	1.03	(.085)
Total	1.27	(.079)	.54	(.087)		

We used a 2 x 2 mixed Ancova analysis with expertise and preparation time as covariates to test whether the mean values of correctness and efficiency differ between query language and complexity, and whether the expected interaction effect occurred.

We found that the main effect of query complexity was significant (F(1,70)=14.01, p=.000), as well as the main effect for language (F(1,70)=4.19, p=.04). We found no interaction effect between language and query complexity.

We found partial similar results for the time spent on the correct and essentially correct result queries of the subjects. The following table shows the estimated mean values for time:

Table 5.8. Time - Complexity

Language	Mean less complex (Sem)		Mean more complex (Sem)		Total (Sem)	
XSLT	676	(144.202)	564	(142.839)	620	(117.582)
XQuery	461	(144.202)	693	(142.839)	577	(117.582)
Total	569	(98.994)	629	(98.059)		

We found a significant main effect for query complexity (F(1,18) = 4.11, p=.06). No main effect for language was found and no interaction effect between query complexity and language was found.

We expected an interaction effect between language and query complexity where the performance differences were smaller for the less complex queries compared to the more complex queries. However, the data in this experiment does not support hypothesis 3. In this experiment, we noticed that for both languages the effectiveness and the efficiency on more complex tasks are lower compared to simple tasks. We also noticed that the correctness rates were lower with XSLT compared to XQuery, but we found no differences for efficiency between the languages. There are several possible explanations why we found different results compared to the previous experiment. First, this may be related to the different levels of expertise of the subjects in the previous experiment and in this experiment. In this experiment we neutralized the effect of user experience by including this as a covariate in the Ancova test. However, this does not mean that the overall performance of subjects in this experiment cannot differ from the overall performance of the subjects in the previous experiment. Second, in this experiment we reformulated the less complex query tasks of the previous experiment. It is possible that the query tasks have become more complex in comparison to the previous experiment. We will elaborate on these explanations in Section 5.4, “Discussion”.

5.3.5. Hypothesis 4: Verboseness

In hypothesis 4 we stated the following:

- *User performance differences between XQuery and XSLT become smaller when verboseness is the only relevant difference between XQuery and XSLT expressions, but there will still be a difference.*

We use Query 1 to verify this hypothesis. See Section 5.2.2.4, "Query 1 - Directed Condition" for an extensive description of this task. To test hypothesis 4, we compare the performance on the query task in a natural condition and in a directed condition. In the natural condition, the XSLT and XQuery expressions may differ in several aspects, e.g. verboseness, Xpath embedding, and expression type. In the directed condition we tried to minimize the influence of aspects other than verboseness. We expect that the performance in the natural condition is determined by all aspects of the query language that is used, whereas the performance in the directed condition is determined by verboseness only. Therefore, we can determine the influence of verboseness on performance by comparing the performance on the following four conditions: for XSLT and XQuery both the natural condition and the directed condition.

In contrast with the previous hypotheses, we only considered the queries of participants who actually tried to solve this query. Otherwise, the results would be biased because all non-attempted queries would count as incorrect. The same performance measurements were used as for the global user performance: correctness of the task and time spent on correct and essentially correct solutions. We provide an overview of the results of the directed and the natural tasks for both languages. See the following table for the correctness:

Table 5.9. Correctness - Query 1

Language	Mean natural (Sem)	Mean directed (Sem)	Total (Sem)
XSLT	.93 (.170)	.71 (.169)	.82 (.119)
XQuery	1.93 (.172)	1.40 (.174)	1.66 (.122)
Total	1.43 (.121)	1.05 (.121)	

A 2x2 between-subjects Ancova analysis was used to test whether these differences were significant. Preparation time and expertise of the subjects were included as covariates. We found a significant main effect for query language ($F(1,68) = 24.32$, $p < .00$) and direction ($F(1,68) = 2.61$, $p < .03$). No significant interaction effect between language and direction was found. This means that the correctness with XQuery was higher compared to XSLT and that the correctness on the directed tasks was lower than on the natural tasks.

See the following table for the time spent on the task:

Table 5.10. Time - Query 1

Language	Mean natural (Sem)	Mean directed (Sem)	Total (Sem)
XSLT	1059 (139.583)	1126 (173.782)	1093 (110.610)
XQuery	310 (119.977)	410 (149.218)	360 (96.369)
Total	685 (91.469)	768 (113.422)	

We used the same 2 x 2 between-subjects Ancova analysis as for correctness, but of course with time as dependent variable instead of correctness. A significant main effect for query language ($F(1,47) = 24.09$, $p < .0005$) was found. No significant main effect for direction, and no interaction effect between language and direction was observed.

For this query, the correctness with XQuery is significantly higher compared to XSLT. In addition, correct queries are solved faster with XQuery compared to XSLT. Therefore, we conclude that XQuery users are more effective and efficient compared to XSLT users on this particular task. Another conclusion is that the direction had a *negative* effect on correctness instead of a positive effect. We expected that the direction would isolate the factor verboseness and that the differences in performance with the languages would become smaller with the directed tasks. This did not happen and instead the direction apparently lead to more complex query tasks for both languages compared to the natural tasks. Because we did not find an interaction effect and because the direction had a different effect than we intended, we cannot accept hypothesis 4. However, the main difference between the solutions with XSLT and XQuery in both conditions seems to be the verboseness of these solutions. Furthermore, there is a significant difference between performance with XSLT and XQuery where the performance with the more verbose language XSLT is lower compared to the more compact XQuery, we consider these performance differences as an indication of the influence of verboseness on performance.

5.3.6. Hypothesis 5: Xpath Embedding

In hypothesis 5 we stated the following:

- *User performance differences between XQuery and XSLT become smaller when Xpath embedding is the only relevant difference between XQuery and XSLT expressions, but there will still be a difference.*

We test this hypothesis with query 2 and query 4 and only the results of subjects who actually attempted these queries were included. The number of subjects who attempted query 2 is 34 for XSLT and 36 for XQuery. Query 4 was attempted by 28 XSLT

subjects and 33 XQuery subjects. We first present the results for query 2, than for query 4.

The following table shows the estimated mean values for correctness of query 2:

Table 5.11. Correctness - Query 2

Language	Mean natural (Sem)	Mean directed (Sem)	Total (Sem)
XSLT	.42 (.182)	.44 (.196)	.43 (.130)
XQuery	1.16 (.178)	1.42 (.180)	1.29 (.126)
Total	.79 (.128)	.93 (.132)	

A 2x2 between-subjects Ancova analysis was used to test whether these differences were significant. Preparation time and expertise of the subjects were included as covariates. We found a significant difference for language ($F(1,64) = 22.38$, $p < .001$). No main effect for direction was found and neither an interaction effect between language and direction. This means that the correctness on this query was higher with XQuery than with XSLT. For the time spent on correct and essentially correct solutions, we found a similar result:

Table 5.12. Time - Query 2

Language	Mean natural (Sem)	Mean directed (Sem)	Total (Sem)
XSLT	1108 (207.971)	1632 (277.470)	1370 (178.993)
XQuery	802 (136.438)	886 (148.719)	844 (102.625)
Total	955 (121.474)	1259 (152.428)	

With the same 2 x 2 Ancova analysis we found a significant main effect for language ($F(1,33) = 5.94$, $p < .02$). No main effect for direction was found and neither an interaction effect between language and direction. We conclude that query 2 was formulated more efficient and more effective by subjects that used XQuery compared to subjects that used XSLT. No effect for the direction or interaction effect between direction and language was found. We discuss these results at the end of this section, but first we proceed with the results for the query 4. The results for the correctness on this query are presented below:

Table 5.13. Correctness - Query 4

Language	Mean natural (Sem)	Mean directed (Sem)	Total (Sem)
XSLT	.27 (.213)	.56 (.230)	.41 (.155)
XQuery	.89 (.200)	.67 (.203)	.78 (.142)
Total	.58 (.146)	.61 (.154)	

A 2x2 between-subjects Ancova was used to test whether these differences were significant. Preparation time and expertise of the subjects were included as covariates. No main effect for direction and no interaction effect between language and direction was found. We did notice a nearly significant difference for language ($F(1,55) = 2.96$, $p < .09$). The correctness rates for XQuery solutions is therefore higher than for XSLT solutions. For the time spent on correct and essentially correct queries we found the following:

Table 5.14. Time - Query 4

Language	Mean natural (Sem)	Mean directed (Sem)	Total (Sem)
XSLT	885 (364.601)	1193 (254.586)	1039 (235.588)
XQuery	747 (185.717)	702 (227.292)	724 (145.336)
Total	816 (199.984)	948 (153.926)	

Again, we used a 2 x 2 Ancova analysis with expertise and preparation time as covariates and we found no main effects for language or direction and no interaction effect between language and direction.

We tested hypothesis 5 with two queries, query 2 (a selection query with multiple criteria) and query 4 (a join query). The results show that there was no effect of the direction; the correctness and efficiency did not differ for in the natural and directed conditions. However, we found performance differences between XSLT and XQuery. For both queries, correctness with XQuery was higher compared to correctness with XSLT. Correct and essentially correct solutions of query 2 were also solved in less time with XQuery. Because we did not find an interaction effect for the language and direction and because there is no effect of direction, we cannot accept hypothesis 5. However, because these specific queries contained a large number of Xpath expressions, we interpreted the higher performance on both queries as an indication that Xpath is easier to use with XQuery compared to XSLT.

5.3.7. Hypothesis 6: Expression Type

Hypothesis 6 is divided in two subhypotheses:

- Hypothesis 6a: *User performance on a specific query is higher with XQuery than with XSLT when a specialized expression for the problem is only available for XQuery.*
- Hypothesis 6b: *User performance on a specific query is higher with XSLT than with XQuery when a specialized expression for the problem is only available for XSLT.*

Hypothesis 6a is tested with query 3 and hypothesis 6b is tested with query 5. We only considered the queries of users that actually had attempted to solve the query. Query 3 was attempted by 33 subjects who used XSLT and 36 subjects who used XQuery. For Query 5 the number of subjects was 30 for XSLT and 32 for XQuery.

The following table shows the correctness of the queries on Query 3:

Table 5.15. Correctness - Query 3

Language	Mean natural (Sem)	Mean directed (Sem)	Total (Sem)
XSLT	1.71 (.173)	1.29 (.178)	1.50 (.124)
XQuery	1.35 (.169)	1.33 (.166)	1.34 (.118)
Total	1.53 (.120)	1.31 (.122)	

With a 2 x 2 between-subjects Ancova analysis that included expertise and preparation time as covariates, we found no main effects, nor interaction effects for correctness. Neither the languages, nor the direction resulted in differences in correctness. For the time spent on correct and essentially correct queries, we found the following estimated mean values:

Table 5.16. Time - Query 3

Language	Mean natural (Sem)	Mean directed (Sem)	Total (Sem)
XSLT	382 (82.720)	465 (102.302)	424 (66.028)
XQuery	536 (83.888)	361 (82.593)	448 (59.122)
Total	459 (58.667)	413 (65.454)	

No main effects or interaction effects were found with a 2 x 2 between-subjects Ancova analysis that included expertise and preparation time as covariates. This means that neither the languages, nor the direction resulted in differences in query solving time for (essentially) correct queries.

We summarize the results for this query as follows. For both the correctness and the time spent on correct queries, no main or interaction effects were found. Therefore, hypothesis 6a is not accepted. We expected a main effect for language where performance with XQuery would be better than with XSLT mainly because of the specialized Xpath 2.0 `distinct-values()` expression. Also, we expected an interaction effect between language and direction where for XQuery the performance would become better in the directed condition and XSLT performance would become worse in the directed condition.

The absence of a main effect for query language is probably because of the low performance with XQuery. When we looked more closely at the result queries that subjects formulated with XQuery, we noticed that most subjects had forgotten to create a list element. Most subjects returned the unique values, instead of a list with elements. This type of error was not made by the subjects who used XSLT. This resulted in a lower overall correctness with XQuery because the result queries were scored as essentially correct. There are two explanations why only subjects working with XQuery made this error. First, subjects became hastily due to the low complexity of the task and the simple solution. Due to hastily reading, it is likely that the subjects using XQuery made an incorrect interpretation of the query task description. We noticed a similar trend in the previous experiment. Second, subjects who worked with XSLT were more used to create XML structures than the XQuery users.

The following table shows the values for correctness of query 5:

Table 5.17. Correctness - Query 5

Language	Mean natural (Sem)	Mean directed (Sem)	Total (Sem)
XSLT	1.03 (.219)	1.15 (.197)	1.09 (.146)
XQuery	.45 (.190)	.47 (.198)	.46 (.137)
Total	.74 (.145)	.81 (.139)	

A 2x2 between-subjects Ancova analysis was used to test if these differences were significant. Preparation time and expertise of the subjects were included as covariates. We found a main effect for language ($F(1,56) = 10.03$, $p < .002$), but not for direction. No interaction effect between language and direction was found. Interestingly, the correctness with XSLT was higher than with XQuery. A similar effect was found for the time that was spent on partial and completely correctly solved queries:

Table 5.18. Time - Query 5

Language	Mean natural (Sem)		Mean directed (Sem)		Total (Sem)	
XSLT	447	(98.556)	469	(91.303)	458	(66.826)
XQuery	596	(156.195)	981	(182.240)	789	(120.730)
Total	522	(92.501)	725	(100.674)		

With the same Ancova analysis as for correctness, we found a significant main effect for language ($F(1,23) = 5.66$, $p < .03$). No significant main effect for direction, nor an significant interaction effect between language and direction was found. This means that correct and essentially correct queries were solved in less time with XSLT compared to XQuery. However due to the lack of an interaction effect between language and direction and because we found no effect of direction, we found no support for hypothesis 6b.

We used two queries to test whether performance improved when using a specialized expression instead of compound expressions: query 3 and query 5. Query 3 is a query that selects unique nodes from the XML document, and the second query was a grouping query. For the first query, we compared the specialized `distinct-values()` function in XQuery with the compound Xpath expression `//elementname[not(.=preceding::elementname)]`. We did not find any differences in correctness and efficiency between the languages, nor between the directed and the natural queries. We did notice that performance with XQuery may have been lower than expected due to a minor error made by all subjects who used XQuery.

With query 5, we compared the specialized `for-each-group` element in XSLT with the compound XQuery clauses `for` and `let`. For this query we found that subjects who used XSLT could formulate this query more effective and efficient compared to subjects who used XQuery. However, we found no differences between natural tasks and directed tasks. When we examined the result queries that subjects formulated, we noticed for XSLT that all subjects used the `for-each-group` element. Apparently, subjects used this element not only for the directed tasks, but also for the natural task. For XQuery we also found little differences between the natural and directed result queries. Therefore, the direction made little difference for the query strategy of both XQuery and XSLT users. Apparently, there was little difference between the natural and the directed condition for this query, but we consider the performance differences on query 5 as a strong indication that the specialized XSLT element leads to a better performance compared to the compound for and/or let operators in XQuery.

5.4. Discussion

In this experiment we tried to validate our findings of the thinking aloud experiment. We accepted the following two hypotheses:

- Hypothesis 1: *User performance on a representative set of queries is higher with XQuery than with XSLT.*
- Hypothesis 2: *User satisfaction is higher with XQuery than with XSLT. Also, user satisfaction of Xpath is higher with XQuery than with XSLT.*

Performance with XQuery was higher compared to XSLT, despite the significant less preparation time that was spent on XQuery compared to XSLT. This shows that the subjects in this experiment learned XQuery faster and performed better compared to XSLT. The results of this experiment must be understood with the following reservations in mind. First, we assumed that the groups of subjects assigned to the four conditions have an equal level of prior knowledge and query language skills. We measured prior knowledge and query language skills with a questionnaire as expertise and preparation time and tested whether the groups had equal levels of expertise and preparation time. Although unlikely, there still is a possibility that some relevant differences between the prior knowledge and query language skills of the subjects were not accounted for in the analyses because of the limited scope of the questionnaire. Second, not the complete syntaxes but only a basic subset of XSLT and XQuery was learned and used by the subjects in this experiment because of the limited available time.

The following hypothesis was not accepted:

- Hypothesis 3: *User performance differences will become larger on more complex queries compared to less complex queries.*

We tried to verify the other hypotheses (4 - 6) by comparing the performance on query tasks with a natural and a directed condition. However, apparently the additional information that was provided to the subjects in the directed conditions did not have the expected effect and this influenced the results of this experiment. For this reason, hypotheses 4 - 6 cannot be accepted. In this section we elaborate on these results and discuss possible explanations for the unexpected results.

We noticed no interaction effects between XSLT and XQuery on less complex queries and more complex queries. This was against our expectations and is, or seems to be, contradicting to the results in our previous experiment. We have two explanations for the unexpected result regarding query complexity in this experiment. First, we changed the query task description of two less complex tasks to test hypotheses about Xpath embedding and expression type. Although these new query descriptions still can be considered as less complex tasks according to the definitions of Reisner and Schlager

(see also Section 2.2.2.5, “Query Types”), they actually may have become more complex than we initially expected. Therefore, the distinction between less complex (selection queries with one or more conditions) and more complex queries (joins and grouping queries) may be too simple to explain the combined effect of user experience, query language, and query complexity on user performance. For instance, we noticed that performance with XSLT on query 5, which is a more complex query, was better compared to the performance on query 3, which is a less complex query. Second, six highly experienced subjects participated in the previous experiment and the overall correctness rates for both languages were nearly 100%. In this experiment, 74 subjects who were generally less experienced participated in the experiment and we noticed that the overall correctness rates were much lower compared to the previous experiment. The different levels of the subject's experience between the two experiments may be another reason for the unexpected differences in performance between less complex and more complex tasks. It seemed that the less complex queries were much harder to solve for the subjects in this experiment compared to the subjects of the previous experiment. Also, for the majority of the subjects in this experiment found it very difficult or even impossible to solve the more complex queries. Therefore, the terms less complex and more complex are probably relative to the experience level of the subjects.

We formulated three hypotheses (4 - 6) about specific language aspects that we wanted to test with two types of query tasks. First, natural query tasks where user performance was determined by all query language aspects at the same time. Second, directed query tasks where user performance was determined by an isolated query language aspect. We expected interaction effects between the performance on the natural and on the directed tasks, but these expected interaction effects were not observed in this experiment. For these three hypotheses, 4 - 6, we summarize the results we found in this experiment as follows:

- Hypothesis 4: *User performance differences between XQuery and XSLT become smaller when verboseness is the only relevant difference between XQuery and XSLT expressions, but there will still be a difference.* This hypothesis was tested with query 1 and performance with XQuery was better than with XSLT on this query task. There was also a significant negative effect of direction: performance was lower in the directed condition for both XQuery and XSLT. We interpret these results as follows: First, the addition instructions and information in the directed condition seemed to have added some unintended complexity compared to the natural query task. Second, the main significant difference between the XQuery and the XSLT expressions in both the natural and the directed condition is, in our opinion, a difference in verboseness. Therefore, we consider the results for this query as an indication that a performance is higher with a less verbose language than with a more verbose language.
- Hypothesis 5: *User performance differences between XQuery and XSLT become smaller when Xpath embedding is the only relevant difference between XQuery and XSLT expressions, but there will still be a difference.* This hypothesis was

tested with query 2 and query 4. On both tasks, we found that performance with XQuery was better than with XSLT. There was apparently no overall effect of direction. Both query tasks contained more Xpath expressions than the other query tasks used in the experiment. Therefore, we interpret these result as an indication that Xpath embedding in XQuery is easier to use and therefore leads to higher performance compared to XSLT.

- Hypothesis 6a: *User performance on a specific query is higher with XQuery than with XSLT when a specialized expression for the problem is only available for XQuery.* This was tested with query 3. However, no performance differences between XSLT and XQuery, and no performance differences between the directed and the natural condition, and no interaction effects were found. Therefore, we do not accept hypothesis 6a.
- Hypothesis 6b: *User performance on a specific query is higher with XSLT than with XQuery when a specialized expression for the problem is only available for XSLT.* This was tested with query 5. We found a significant effect for language where performance with XSLT was better compared to performance with XQuery. There was no effect of direction, nor an interaction effect between language and direction. We interpret this result as an indication for the correctness of our hypothesis because this is the only query task were performance with XSLT was higher than with XQuery and this was also the only query task were XSLT, in contrast to XQuery, included a specialized operator.

In general, we noticed that there was no effect of direction, except for query 1 where the direction had a negative effect on performance. The general trend for language was that XQuery performance is overall better, except for query 5 where performance with XSLT was better and query 3 where no performance difference between XQuery and XSLT were observed. We expected that the direction would have an overall positive effect on the subject's performance because subjects were provided with additional instructions about which expressions they should use and/or partially standard query templates. However, the expected overall positive effect did not occur. Instead, direction had no effect, or a negative effect on performance.

It seems unlikely that, for instance, providing a nearly complete template of a query has no influence on the performance of that task. Therefore, we assume that the additional information provided in the directed conditions was not helpful, but on the contrary may have added complexity for some subjects. We illustrate this with the correctness scores on query 4. For this query, we noticed a main effect of language where on both conditions correctness with XQuery was higher than with XSLT. To gain a better understanding of the effect of direction on this query task and on the correctness of subjects with different levels of expertise, we re-analyzed the data of this query. Instead of a 2 x 2 between-subject Ancova, we used a 2 x 2 x 3 between-subject Ancova test with language, direction, and subject's expertise (3 levels) as independent variables. With this analysis we found, of course, a nearly significant main effect for language where correctness with XQuery was better compared to correctness with XSLT ($F(1,48)=3.68$, $p < .06$). We also found a main effect for

expertise, where subjects with high expertise performed better compared to subjects with medium and low expertise ($F(2,48)=3.67$, $p < .03$).

In addition, two interaction effects were found. First, we noticed an interaction effect between expertise and direction ($F(2,48)=4.20$, $p < .02$). When we compared the correctness rates of subjects with low, medium, and high expertise on both the natural and the directed queries, we noticed the following. Subjects with low expertise performed better with the directed queries, subjects with medium expertise scored approximately the same on directed queries, and subjects with high expertise performed worse on directed queries. Second, we noticed that there was also an interaction effect between language, expertise, and direction ($F(2,48)=4.47$, $p < .02$). When we compared the correctness rates on these conditions, we noticed that the interaction effect between expertise and direction occurred only with XQuery.

For query 4, the direction has a negative effect on the performance of high experienced participants and in particular for XQuery, whereas it has a positive effect on the performance of low experienced participants. To find an explanation for this unexpected result, we looked at the queries of the participants and noticed the following. For the subjects with high expertise in the non-direction XQuery condition, *none* of the participants formulated a query that had a structure that was similar to the template given in the directed condition. All highly experienced subjects in the non-directed XQuery condition ($N = 5$) were able to solve the query correctly, whereas only one participant in the directed condition created a partial correct query ($N=4$). This indicates that the template we provided was counter intuitive and added additional complexity to the task for the experienced participants. For XSLT no differences were found. Apparently, a join query is difficult with XSLT and the template had no influence on the task complexity. This additional analysis shows that the direction on the join query had more complex effects than we intended. It is likely that similar effects occurred with the other directed query tasks in this experiment.

5.5. Conclusions

We summarize the results of this experiment as follows. First, the overall performance with XQuery was higher compared to performance with XSLT on the set of five query tasks in this experiment. Second, user satisfaction of XQuery was higher compared to XSLT and also Xpath satisfaction was highest with XQuery. Third, contrary to our expectations, there was no interaction effect between task complexity and query languages. This is probably due to small changes we made to the less complex query tasks from the previous experiment. Another reason may be that the global distinction between less complex and more complex is not sufficient to characterize the level of complexity of query tasks and the differences in level of expertise between the subjects of the current and the previous experiment. Fourth, we conclude that we were not completely successful with our attempt to isolate particular language aspects with the directed query tasks. We were not able to measure the performance that is caused by

an isolated language aspect, because the directions introduced some unintended and uncontrolled characteristics of the query tasks. Therefore, the use of query templates and the enforcement of a particular query strategy or query expression needs more attention to gain a better understanding of the influence of language aspects on user performance. The differences between performance with XQuery and XSLT on the natural set of tasks used in the experiment suggest that these differences are mainly related to differences in verboseness, expression type, and Xpath embedding. Fifth, the results suggest that the subject's level of expertise has a large influence on performance with these query languages.

Chapter 6. Experiment 3: Explanation

6.1. Introduction

In the previous chapter we described an experiment in which we tested several hypotheses related to usability of XML Query language. We found that both performance and satisfaction with XQuery was higher compared to XSLT. We also found strong indications for the influence of the following factors: verboseness, Xpath embedding, and expression type. More specifically, we found indications that XSLT's verboseness and method of Xpath embedding had a negative effect on the user performance. We also noticed that performance with XSLT on a grouping query was better compared to XQuery because of XSLT's specialized for-each-group element. Another conclusion of the previous experiments was that user experience has a large effect on performance with XML query languages.

However, some cautiousness is necessary when interpreting the results of this experiment. First, we included five query tasks in the experiment and based on previous research, we distinguished between less complex and more complex query tasks. However, we noticed that this distinction between less complex and more complex tasks was not suitable for the queries we selected. We concluded that this binary distinction was too coarse to represent the concept of task complexity. In addition, the number of and the differences between the query tasks may have been too limited to represent all normal XML query tasks. Second, related to the previous item is the following. We wanted to test if subjects were capable of solving a particular query problem. To test this, we provided subjects with a query task description that was related to an XML document. During the rating of the subject's result queries, we noticed that some errors may have been caused by an incorrect interpretation of the query task description instead of problems related to solving the query problem. Therefore, we may have induced a wrong query task interpretation, instead of exclusively testing the subject's capabilities to solve a particular query problem.

In this experiment we want to further verify the findings of the previous experiments and to explain these findings in more detail. In addition, we want to investigate the influence of XML document structure on the performance with XSLT and XQuery. Based on the previous experiments, the following factors are likely to influence the user performance with a query language. First, the complexity of a query task. Instead of making a binary distinction between less and more complex we will use a scalar value in this experiment. Second, the experience of the user. The user's experience will be assessed with a questionnaire and by considering the academic achievements of the subjects. Third, the verboseness of the query language which is represented by

the number of characters that are used in a query expression. Fourth, embedding of Xpath expressions. This factor is actually not distinguishable from the query language that is used. Therefore, we assume that differences due to differences in Xpath embedding between XSLT and XQuery become more obvious when more and lengthier Xpath expressions are used in query language expressions. For this reason, the number of Xpath expressions, and the total length of Xpath expressions in a query are used to represent Xpath embedding in XSLT and XQuery. Fifth, expression type seemed to be an important factor in the previous experiment. In this experiment we do not want to direct subjects towards a particular query solution. Therefore, we cannot be sure which expressions subjects will use. We assume that the number of expressions in a query is lower when a specialized expression is used. For this reason, we count the number of expressions in a query because we consider this as a more general notion to distinct between compound expressions and specialized expressions. Sixth, in this experiment we include two types of XML documents to test whether data structure influences user performance with XSLT and/or XQuery. An important difference with the previous experiment is the design of the experiment and the statistical methods to test our research questions. In the previous experiment we manipulated independent variables, e.g. verboseness, and tested for differences of the values of dependent variables, e.g. correctness, in the different conditions. In this experiment we measure the natural occurring values of certain predictors, e.g. verboseness of a query language, on a large set of query tasks. We test if, and to what extent a set of predictors can explain the values of the dependent variable, e.g. correctness, on a set of tasks. In the following section we state the research questions for this experiment.

6.2. Research Questions

In this experiment we want to answer the following questions:

1. Which XML query language facilitates the highest user performance: XSLT or XQuery? We expect that XQuery will have a better performance because the queries formulated with this language are generally shorter, and the method that is used to embed Xpath expressions is simpler compared to XSLT. Also, user performance in the previous experiment was higher with XQuery than with XSLT on five query tasks.
2. Which XML query language leads to the highest satisfaction: XSLT or XQuery? For the same reasons as stated above, we think that the satisfaction of XQuery will be higher.
3. What are the most important predictors for performance with XSLT and XQuery? Are these predictors the same for both languages? We consider the following potential predictors:
    a. Complexity of the query task, or shortly task complexity;
    b. Experience of the user;
    c. Verboseness of the query language;
    d. Embedding of Xpath expressions;

e. Number of expressions or elements.
f. XML document type;

In the following section we discuss how we will investigate these research questions.

6.3. Method

6.3.1. Subjects

The subjects in this experiment were 33 Bachelor and Masters students Information Science. A large number, 27 Bachelor students, also participated in the previous experiment. The remaining 6 subjects were Master students and participated in an earlier edition of the course 'Exchange Languages'. Participation was voluntary and the subjects received a fee of 15 Euro.

To determine the experience of the subjects with XML query languages related techniques and languages (databases, XML, HTML, imperative and functional programming languages, and so on) the following sources were used:

1. a questionnaire;
2. the academic achievements of the subjects.

The questionnaire was very similar to the questionnaire used in experiment 2 (see Appendix B, *Questionnaire - Prior Experience*). Most questions were duplicated from experiment 2, only the queries that were related to specific circumstances of the introductionary XML course were replaced with questions related to the subject's experiences with XML outside the university's curriculum. With the questionnaire and the academic achievements of the subjects, we could detect possible differences in the prior relevant knowledge and skills between the XQuery and XSLT groups. This was necessary because prior knowledge and skills are likely to influence user performance and therefore it is desirable to check whether the groups are not significantly different in this respect. Besides the academic achievements of the subjects, we used a questionnaire to determine prior relevant knowledge and skills of the subjects. This was done because we were not allowed to ask for the student's academic achievements at the faculties office without permission of the subjects. See for the results Section 6.4.1, “Prior Experience of the Subjects”

6.3.2. Materials

6.3.2.1. Query Languages

XQuery 1.0 and XSLT 2.0, as implemented by Saxon 7.8 [Saxon]. This is the same query processor as used in experiment 2.

6.3.2.2. Query Tasks

The performance of XQuery and XSLT needs to be examined on a large, representative set of queries. Therefore, we selected 64 query tasks from multiple sources. Our criterion for selection was that the query task must be suitable for the general level of expertise of our subjects, that is, the type of query could have occurred in the context of the course "Exchange Languages". This means that queries that we assessed were too complex for the current level of expertise of the subjects were not selected for our experiment. The queries were applied to two types of documents, namely a document-oriented XML document and a data-oriented XML document. The document-oriented XML document is based on sgml.xml from the W3C XML Query Use Cases: Use Case "SGML": Standard Generalized Markup Language. [W3C-QU]. The data-oriented document is based on prices.xml from the W3C XML Query Use Cases: Use Case "XMP": Experiences and Exemplars, Query 10. [W3C-QU]. We selected queries from the following sources:

- W3C XML Query Use Cases [W3C-QU].
- W3 School XQuery tutorial [W3-S].
- Xmark Benchmark [SWK+02].
- Xbench family of benchmarks, text-centric multi-document collection and the data-centric single-document collection [YOK02].
- The queries from our previous experiment.

After we selected the queries, we established for each query the relevant properties as follows. First, we created for each query one or more correct solutions in both XSLT and XQuery. Of course, for each query a large number of different solutions can be made, but we tried to find only natural and likely solutions and we checked the likelihood of a particular solution both with the original solution given in the source, as well as with the result queries from our previous experiment. Second, we established for each query the properties of the solutions and this was done as follows:

1. *Complexity:* we counted the number of elements, attributes and the values that should be returned with the query, as well as the number of conditions that were used in the query. If the value of an element or attribute of a subquery was used in a condition, than these values were counted twice to represent the complexity of subqueries in a query. The complexity of a query is independent of language and document schema.
2. *XML Document type:* we used two types of XML documents:
   a. A document-oriented XML document, see Appendix F, *Document-oriented XML*.
   b. A data-oriented XML document, see Appendix G, *Data-oriented XML*.

   We created a solution for each type of document, so each query was applied to both types of documents.

3. *Number of expressions:* we counted for each query and each language the number of XQuery keywords and the number of XSLT elements respectively. For XSLT queries we excluded the element xsl:transform (or xsl:stylesheet) because this is a standard element for each XSLT query. Most queries had multiple solutions, and in that case we took the mean number of expressions.
4. *Verboseness:* we counted the number of characters in each query for each language after removing the tabs, line ends and double spaces from the expressions. We used the mean values when multiple solutions were made.
5. *Xpath embedding:* we counted for each solution and for each language:
    a. Number of Xpath expressions: that is, the total number of Xpath expressions embedded in an XQuery or XSLT query;
    b. Xpath length: that is, the total number of characters in Xpath expressions embedded in an XQuery or XSLT query.

The number of XSLT elements, XPath expressions and the length of Xpath expressions were determined automatically. For XQuery, the number of expressions was determined manually, the number of Xpath expressions were determined automatically by counting the character '$' in each query. The Xpath expressions were manually selected from the queries, the number of characters was determined automatically.

In Example 6.1, “Determining the Properties of Queries” we provide one of the 64 query tasks from our experiment, together with the source of the query, the possible solutions for XQuery and XSLT and the operationalization of the relevant factors. The queries from the experiment can be found in [Gra05].

Example 6.1. Determining the Properties of Queries

The source of the following query task is the first query from the W3C Use Case "XMP": Experiences and Exemplars, Query 1 [W3C-QU]. For the Text XML document that is used in the experiment, we reformulated the query task as follows:

Query Task. Select the chapter that has a number higher than 1 and with title "Getting to know SGML", include the number of the chapter as attribute and the intro of this chapter.

We used the following solutions:

XQuery Solution 1.

```
for $t in //chapter
where $t/@number > 1 and $t/title="Getting to know SGML"
return
<chapter number="{$t/@number}">
 {$t/intro}
</chapter>
```

XQuery Solution 2.

```
for $t in //chapter[title="Getting to know SGML" and @number > 1 ]
return
<chapter number="{$t/@number}">
 {$t/intro}
</chapter>
```

XSLT Solution 1.

```
<xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="2.0">
<xsl:template match="text()" />
<xsl:output method="xml" indent="yes" />

<xsl:template match="chapter[@number &gt; 1]
 [title='Getting to know SGML']">
   <chapter number="{@number}">
    <xsl:copy-of select="intro" />
   </chapter>
</xsl:template>
</xsl:transform>
```

XSLT Solution 2.

```
<xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="2.0">
<xsl:output method="xml" indent="yes" />
<xsl:template match="report">
   <xsl:for-each select="chapter">
 <xsl:if test="@number &gt; 1 and title='Getting to know SGML'">
      <chapter number="{@number}">
   <xsl:copy-of select="intro" />
    </chapter>
   </xsl:if>
</xsl:for-each>
</xsl:template>
</xsl:transform>
```

XSLT Solution 3.

```
<xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="2.0">
<xsl:output method="xml" indent="yes" />
<xsl:template match="//chapter[title = 'Getting to know SGML']">
<xsl:if test="@number > 1">
<chapter>
<xsl:attribute name="number">
<xsl:value-of select="@number"/>
</xsl:attribute>
<xsl:copy-of select="intro" />
</chapter>
</xsl:if>
</xsl:template>
</xsl:transform>
```

The relevant properties for this query task are established as follows:

1. *Complexity*: the complexity of this query task is the sum of the report and the condition items. Items are elements, attributes and their values. Values from subqueries are counted twice to model the complexity of subqueries in a query. The following elements and attributes are counted:
   - The number of report items is 3, namely 1 chapter element, 1 number attribute and 1 intro element.
   - The number of condition items is 4, namely 1 title with 1 value "Getting to know SGML" , 1 number attribute with 1 value higher than 1.

   The complexity of this query task is therefore 7.
2. *Data structure:* this query is defined for the document-oriented XML document.
3. *Number of expressions:*
   - XQuery: XQuery Solution 1 has 4 keywords (for where and return), XQuery Solution 2 has 3 keywords (for and return). Therefore, the number of expressions for XQuery on this task is 3,5.
   - XSLT: XSLT Solution 1 consists of 4 XSLT elements (2 x xsl:template, 1 x xsl:output, 1 x xsl:copy-of), XSLT Solution 2 of 5 XSLT elements and XSLT Solution 3 of 6 XSLT elements. Therefore, the number of expressions for XSLT is: (4 + 5 + 6) / 3 = 5.
4. *Verboseness:*
   - XQuery: the XQuery Solution 1 consists of 138 characters, XQuery Solution 2 consists of 127 characters. Therefore, the verboseness (the mean length) of this query for XQuery is 132,5.
   - XSLT: the XSLT solutions have 329, 373, and 388 characters respectively. Therefore, the verboseness (mean length) of this query for XSLT is 363,3.
5. *Xpath embedding:*
   - XQuery: XQuery Solution 1 has 5 Xpath expressions with a total length of 72 characters:
     - //chapter
     - $t/@number > 1
     - $t/title="Getting to know SGML"
     - $t/@number
     - $t/intro

     XQuery Solution 2 has 3 Xpath expressions with a total length of 68 characters:
     - //chapter[title="Getting to know SGML" and @number > 1]
     - $t/@number
     - $t/intro

     Therefore, the XQuery solutions have a mean number of XPaths of 4 and a mean length of 70.
   - XSLT: the XSLT solutions have 4, 5, and 4 Xpath expressions respectively, with a total length of 64, 65, and 67. Therefore, the XSLT solutions have a mean number of XPaths of 4,33 and a mean length of 65,33.

See [Gra05] for an overview of all the queries in the experiment.

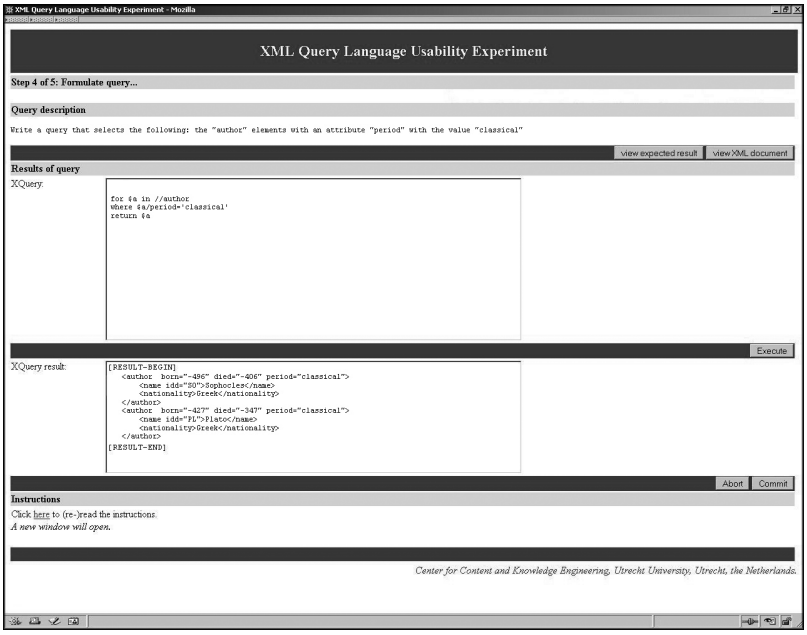
6.3.2.3. Experimental Platform

We used the same experimental platform as in the previous experiment: the XML Testbed (TERS) [VZw04]. One major change was made to the interface to overcome the potential problems that subjects may have with query tasks interpretation. Instead of providing the subjects only with the query task description, we also provided the expected result data that the subjects should retrieve. The expected result data for each query task could be viewed by the subjects by pressing a button. The query interface consisted of the following fields and pages:

1. Welcome and registration page;
2. General questionnaire about the subject's experience;
3. Drop down list which contained the query tasks that were assigned to the registered subject;
4. Query interface that consists of the following:
    a. Query task description and the reference to the related XML document. For instance: *Find all names. Document = report.xml;*
    b. A button which opened the related document when pressed on;
    c. A button which opened the expected result of the query task. This feature was added to reduce the possibility of misinterpretation of the query task description;
    d. A text field in which the subject was supposed to enter the correct query statement, either with XQuery or with XSLT, depending on which group the subject was assigned to;
    e. A button "execute" that submitted the query statement to the query processor.
    f. A text field in which the result of the query or the error messages appeared after pressing the button "execute".
    g. A button "abort" that the subject was supposed to use if he wanted to pause or when he wanted to try another query. After pressing "abort", the subject returns to the drop down list with queries. The aborted query still remains in the list and when the subject choose this query again, the aborted query and the last result is shown again.
    h. A button "commit" that the subject was supposed to use if he thought that the query was correct. After pressing "commit", the subject returns to the drop down list with the query tasks and the committed query task is removed from the list.
5. Questionnaire about the satisfaction of the subject about the query language he or she has used.
6. A "Thanks for the participation" page.

Here is a screen shot of the query interface:

Figure 6.1. Screenshot of the Query Interface



6.3.3. Procedure

Subjects performed query tasks with the same language they worked with in the previous experiment. Subjects who did not participate in the previous experiment were asked which language they preferred: XSLT or XQuery. The XSLT group consisted of 19 subjects, the XQuery group consisted of 14 subjects.

The 128 query tasks (64 queries and two document types) were randomly ordered and assigned to the subjects. Approximately 12 - 18 query tasks were assigned to each subject. The random ordering and assignment was performed for two reasons. First, we wanted to compensate for learning effects. Second, we wanted to obtain a score for performance on each task that is more or less independent of individual subjects. This is done by assigning multiple subjects to each query task and by calculating the mean performance of these subjects on each task. In this way, we obtain a more objective and reliable user performance on a large set of different query tasks. The random assignment procedure was as follows:

1. In MS Excel a column with 128 randomly generated numbers between 1 and 128 was created.
2. In the second and third column the task numbers (1 to 64) and the document type (1 and 2) respectively were placed. For each task, two document types were used.

3. This worksheet was sorted by the first column, so the task and document columns were now randomly ordered.
4. Two new columns were created: XQuery and XSLT.
5. In a separate worksheet 128 random number between 1 and 35 were generated: the id values of the subjects.
6. The randomly generated subject id's were one by one copied in the XSLT or XQuery columns, depending on which group the subject was placed.
7. Steps 5 and 6 were repeated until 12-18 tasks were assigned to each subject. After a subject was assigned to an appropriate number of assignments, the subject's id was removed from the randomly generated id's.
8. The subject id's and the accompanying task and document numbers were copied into a new worksheet. If a task was assigned to the same subject twice, the second task was replaced with the task number + 1.
9. The tasks and documents were renumbered: tasks of the document type 1 remained unchanged (1-64), tasks of document type 2 were replaced with the task number + 64. For instance, task 1 with document 2 was replaced with task 1 + 64 = 65.
10. Every subject was assigned to 12 - 18 tasks of either XSLT and XQuery with a value between 1 and 128.

Each subject was assigned to a set of 12 - 18 query tasks, where the number of tasks varied because of the method of random assignment. For each subject, his or her unique set of tasks was placed in a drop down list in the XML Testbed interface. To compensate for learning effects, subjects were asked to select and complete the tasks in the random order they were presented. However, they were allowed to skip a query task and continue with the following task if they thought that the task was too difficult to solve. Note that as a result of this setup, most query tasks were executed by multiple subjects, but some were done by only one subject and a few were done not at all. See Section 6.3.2.3, “Experimental Platform” for an example how the query tasks are presented to the subjects.

The experiments were conducted in one of the students' practice rooms of the Information & Computing Sciences Institute. Students worked on regular PCs with MS Windows XP and MS Internet Explorer. The amount of subjects varied from 1 - 8 per session. Each session lasted three hours:

1. 5 minutes were used to fill in the questionnaire about the subject's experience.
2. 40 minutes were used to refresh the subject's knowledge about the query language, the query interface and the reference materials. This was done by handing the subjects a 5 page introduction that consisted of a short introduction to the interface, references to the material, and a number of example queries including the correct solution. They also received the DTD and a print of the XML documents that were used during the experiment. The example queries were related to another XML document, but could be executed in the interface. Subject's were allowed to ask any question about the language, interface and materials.

3. Two hours of query solving. During the query solving phase, subjects were only allowed to ask questions that were related to the interface. If the subjects were not finished with all the queries at the end of the session, they were asked to reopen the aborted queries and commit these as well. This was necessary because we considered an aborted query task as a query that was a failed attempt of the subject to solve the query.
4. 5 minutes to fill in the questionnaire about the subject's satisfaction of the language used.
5. 10 minutes spare time that can be used for a coffee break, compensation time for subjects who arrived late, and so on.

6.3.4. Design and Analysis

In this section we discuss how the concepts that are likely to influence the performance with XML query languages are operationalized in predictors and which statistical methods are used to answer the research questions.

Task and Query Language related Predictors. In Section 6.3.2.2, "Query Tasks" we discussed how the concepts complexity, verboseness, number of expressions, and Xpath embedding are operationalized in properties of XML query language expressions. We will provide a short overview of how these concepts are included in our analyses as expected predictors:

1. Task complexity is defined as the sum of elements, attributes, and their values required in the conditions and or the result of the query task. This predictor is independent of XML query language.
2. Verboseness is the mean number of characters of possible, and reasonable, query language expressions.
3. Number of expressions is defined as the number of keywords in XQuery and the number of XSLT elements excluding the xsl:transform element in XSLT. The number of expressions is a generalization of expression type from the previous experiment.
4. Xpath embedding is defined by two predictors:
   a. Number of Xpath expressions: i.e., the total number of Xpath expressions embedded in an XQuery or XSLT query;
   b. Xpath length: i.e., the total number of characters in Xpath expressions embedded in an XQuery or XSLT query.

Besides these predictors that are assessed by query task and the query language, we also include predictors that are related to prior experience of the user.

User related Predictors. We used two sources to determine the prior experience of the subjects. First, a questionnaire that was provided at the beginning of the experimental session. Second, information provided by the office of the faculty about

the academic achievements of the subjects. Both sources are complementary to each other in some aspects. For instance, subjects are asked in the questionnaire whether they participated in certain programming courses while this information is also available when looking at the academic achievements. However, due to privacy issues we were obligated to ask for the subject's permission to include their academic achievements in the experiment and therefore we included some complementary questions in the questionnaire. The experience of the subjects that we computed with the subject's answers on the questionnaire has therefore an overlap with the other, more detailed, information regarding experience of the subjects. For hours of XSLT and XQuery experience we combine information from both sources. The academic achievements were used to determine the hours of experience with XSLT and XQuery during the study and the questionnaire was used to determine whether the subjects had experience with these languages in other contexts. The predictors represented below are the aspects of prior experience of the subjects that we included in our analyses:

1. Hours of XSLT experience, based on both the questionnaire and academic achievements;
2. Hours of XQuery experience, based on both the questionnaire and academic achievements;
3. Number of participated courses during the study, based on academic achievements;
4. Average score on all participated courses, based on academic achievements;
5. Number of participated programming language courses during the study, based on academic achievements;
6. Average score on all participated programming language courses, based on academic achievements;
7. General experience of the subjects, based on Appendix B, *Questionnaire - Prior Experience* (the mean values of questions 1-21).

For every query task we determined the value of the predictors mentioned above. The values of the subject related predictors were determined by computing the mean values of the subject related predictors that attempted the query. For instance, a query task is attempted by subjects with an average of six hours XQuery experience if three subjects with respectively four, six, and eight hours of XQuery experience attempted to solve this task. In a similar manner we determined both the average correctness and the average time for every query task for both XSLT and XQuery.

Performance. In this experiment, we will test to which extent these tasks, query language, and subject related variables can predict the correctness of, and time spent on, a set of query tasks. The actual time spent on a query task is registered automatically by the XML Testbed system. The correctness of the result queries is determined by the experimenter and scored with '0' (incorrect) or '1' correct. In the previous experiment we also used a rating 'essentially correct' to score the result queries of subjects who had solved the most difficult part of the query task but made a minor error. This may have been caused by an incorrect interpretation of the query task. We dropped the rating 'essentially correct' to make the scoring of the result queries more clear and to

leave no room for subjective judgments of the result queries. In this experiment, we provided the expected result data to the subjects. The change that a subject misunderstands a query task and creates an incorrect query is than greatly reduced. For instance, we ask the subject to retrieve a list of elements with a value lower than 3 and we provide the following expected result data:

```
<list>
 <item>0</item>
 <item>2</item>
 <item>1</item>
</list>
```

The subject knows precisely which data should be retrieved and remaining errors cannot be caused by an incorrect interpretation of the query task description. If a subject now returns only the values without the elements, it is likely that the subject, for instance, cannot create elements.

Statistical Tests. According to our theory, the performance (correctness and time) on a set of query tasks can be explained with the predictors mentioned above. We used multiple linear regression analysis to test which predictors can explain the largest amount of variance in performance with XSLT and XQuery on a set of tasks. This statistical method is similar to Anova because they both explain the variance on the values of a dependent variable, like correctness, based on the values of one or more independent variables, like task complexity and verboseness of the query language expression. However, in multiple regression analysis independent variables are not manipulated by an experimenter but instead the natural occurring values of independent variables are measured in order to see whether they predict the score on the dependent variable. The results of multiple linear regression are interpreted as follows. First, the value of multiple R Square (R^2) represents the proportion of the variance in the dependent variable which is accounted for by the model. For instance, if we consider the influence of the predictors mentioned above on the correctness rate with a query language we may find a multiple R^2 value of .50. This means that the significant predictors in the model have accounted for 50% of the variance for correctness of the query tasks. Second, the importance of a predictor in the model is expressed in the Beta (the standardized regression coefficients). For instance, we may find two significant predictors for correctness with XSLT, namely complexity with a Beta value of -.500 and verboseness with a Beta of -.400. A negative Beta value indicates that the higher the value of the predictor, the lower the correctness rate will be. In this example, complexity has a larger value than verboseness and therefore has complexity a greater impact on the correctness than verboseness. There are several methods in SPSS how predictor values can be entered in the regression model, the most common methods are enter and stepwise. With the method enter, all predictors are included in the regression model and contribute to the proportion of the variance in the dependent

variable. With the method stepwise, the predictors are added to the regression model one by one. Predictors are only added to the model when they significantly contribute to the proportion of variance of the dependent variable. The stepwise method results in a more efficient regression model compared to the method enter, because only the best subset of predictors are included in the model. On the other hand, the method enter will result in higher R Square values. We used the method stepwise because this method results in the model with the smallest subset of predictors.

6.4. Results

6.4.1. Prior Experience of the Subjects

Before we analyzed our data to answer the research questions, we checked for differences in the prior relevant knowledge and skills between the XQuery and XSLT groups. We asked subjects if we were allowed to use their study results in our analysis. One subject did not allow it and therefore in the XSLT group the number of subjects is 18 for the academic achievements, and 19 for the subject experience that is computed based on the questionnaire. We found the following mean values and standard deviations for the XSLT and XQuery group:

Table 6.1. Experience of the Subjects

	XQuery (n=14) Mean (Stdev.)	**XSLT (n=18) Mean (Stdev.)**
Hours of XSLT experience	11.9 (2.4)	16.7 (4.4)
Hours of XQuery experience	7.0 (2.5)	4.1 (3.4)
Number of courses	11.6 (7.8)	20.3 (10.3)
Average score on courses	6.1 (.9)	6.7 (.9)
Number of programming courses	4.2 (4.6)	5.5 (3.0)
Average score programming courses	6.6 (1.0)	6.9 (1.3)
General experience of the subject	3.2 (.9)	3.6 (.7) (n=19)

In general, the experience of the group of XSLT subjects tends to be higher compared to the XQuery group, except for hours of XQuery. With independent samples t-tests we found significant differences for the following variables:

1. Hours of XSLT experience (t = -3.98, df = 27,38, p = .000, two-tailed). The variances of both groups are not equal.
2. Hours of XQuery experience (t = 2.70, df = 30, p = .011, two-tailed)
3. Number of courses (t = -2.66, df = 30, p = .012, two-tailed)
4. Average score on courses (t = -1.73, df = 30, p = .095, two-tailed)

This means that the XSLT group, compared to the XSLT group, had more hours of XSLT experience, attended more courses and had a higher average score on these courses. The XQuery group had more hours of XQuery experience. The difference between the hours of XSLT and XQuery experience can be explained by the participation of 27 subjects with the previous experiment. These subjects worked with the same language as in the previous experiment, so the subjects in the XQuery group in this experiment naturally have more experience with XQuery compared to subjects in the XSLT group. The reason for the difference in average number of attended courses, is that the six master students who participated in the experiment were assigned to the XSLT group. This was necessary because these students had experience with XSLT and not with XQuery. Of course, the number of courses attended by the master students is higher compared to the bachelor students. The explanation why the XSLT group has a higher average score on the attended course is less clear. Possible explanations might be: students perform less at the beginning of their study compared to the end of the study, or students who studied longer had more opportunities for re-examinations.

6.4.2. XSLT and XQuery Performance

The first research question we try to answer with this experiment is whether XQuery performance is better than XSLT performance. The performance on the query tasks is defined in terms of:

1. Correctness of the query: is the result of the query the same as the expected result data that is provided to the subjects and according to the query description? The correctness for each query is assessed by the experimenter.
2. Time of the query process: this is the time that a subject needed to solve a query task, which is the time between the selection of the query and the moment the subject presses the abort or commit button. After pressing abort, when a query is re-selected, the time of the previous attempt is added to the current attempt.

64 query tasks were defined on two types of documents, therefore the number of assignments in the experiment is 128. We first analyzed our data with the 128 assignments, but we noticed that the factor document type did not appear in any model as a significant predictor. See Table H.3, “Predictors for correctness” and Table H.4, “Predictors for time” for these results. In other words, differences in performance were not related to the difference between the text-oriented or the data-oriented XML document. Therefore, we can safely exclude this factor from our analysis and combine

the assignments with the same underlying query task to one assignment. The data for each query becomes more consistent and reliable because the number of subjects that attempt a particular query increases. In the following sections we show the results when the queries are grouped regardless data structure. See Appendix H, *Results for Assignments* for the results of the analysis on the 128 assignments.

Most query tasks were performed by multiple subjects, and we defined the performance on a particular query task as the average of the subject's performance on this particular assignment. We noticed that the mean number of subjects per task was 3.3 for the XSLT group and 3.2 for the XQuery group. A dependent samples t-test showed us that this difference was not significant: $t(63) = .80$, $p = .425$.

For each query task we computed the average correctness and average time spent on the task for both XSLT and XQuery. In the previous section we noticed that the subjects in the XSLT group where more experienced in some aspects compared to subjects in the XQuery group. Therefore, we used an Ancova to compare the mean values of correctness and time for XSLT and XQuery. We included the measurements of experience as covariates in the analysis. The results are discussed in the following sections.

6.4.2.1. Correctness

The total number of query tasks in this experiment was 64. During the experiment it became clear that one particular query task was not correctly formulated by the experimenter and therefore we excluded this tasks from the analysis. One task was not attempted by any subject in the XSLT group and two tasks were not attempted in the XQuery group. Therefore, the number of tasks is 62 with XSLT and 61 with XQuery. The table below shows the estimated mean values and the standard error of the mean (Sem) for correctness. The estimated mean values are corrected for the influences of the following factors: hours of XSLT experience, hours of XQuery experience, number of courses, average score on courses, number of programming courses, average score on programming courses, general experience of the subjects.

Table 6.2. Correctness

Language	Mean	(Sem)	N
XSLT	.62	(.061)	62
XQuery	.78	(.062)	61

An analysis of covariance showed that this difference is not significant: $F(1,114) = 2,15$, $p < .15$.

6.4.2.2. Time

The time that a user of a query language needs to formulate a query language expression is a measurement of the user's efficiency with that query language. In the previous experiment, we considered the time that was spent on correct and partially correct query language expressions. This was necessary because incorrect query language expressions can be solved both in a short time and in a long time. So, time is a meaningful measurement for incorrect queries. In this experiment, we only included the average time spent on a query task when the average correctness of that query tasks was higher than zero. The table below shows the estimated mean values and the standard error of the mean (Sem) for time. The estimated mean values are corrected for the same influences as for the analysis of correctness: hours of XSLT experience, hours of XQuery experience, number of courses, average score on courses, number of programming courses, average score on programming courses, general experience of the subjects.

Table 6.3. Time

Language	Mean	(Sem)	N
XSLT	574	(73,696)	57
XQuery	330	(72,640)	58

An analysis of covariance showed that this difference is nearly significant: $F(1,106) = 3,359$, $p < .07$.

6.4.2.3. Summary

We have seen that the time spent on at least partial correct queries is higher with XQuery compared to XSLT. No significant difference was found for the correctness rates between XQuery and XSLT.

6.4.3. Satisfaction

The second research question we try to answer with this experiment is whether satisfaction of the user is higher with XQuery compared to XSLT. Satisfaction is measured with the same questionnaire used in the previous experiment. See also Appendix C, *Questionnaire - Satisfaction*. The only difference is that some questions were removed, i.e. questions related to specific Xpath queries that were included in experiment 2 but not in experiment 3. The table below shows the estimated mean values for satisfaction for subjects that used XSLT and subjects that used XQuery:

Table 6.4. Satisfaction

Language	Mean (Sem.)	N
XSLT	4.03 (.210)	18
XQuery	4.68 (.248)	14

The estimated mean values are corrected for the same influences as for the previous analysis: hours of XSLT experience, hours of XQuery experience, number of courses, average score on courses, number of programming courses, average score on programming courses, general experience of the subjects. An analysis of covariance showed that the difference between the satisfaction of XSLT and XQuery is nearly significant: $F(1,23) = 3.02$, $p < .10$.

6.4.4. Performance Predictors

The third research question we try to answer is: what are the most important predictors for performance with XQuery and XSLT and are these predictors the same for both languages?

As mentioned in Section 6.3.4, "Design and Analysis", stepwise multiple regression analysis was performed to explain the variance in our performance measurements: correctness and time. The following task and language related predictors were included in the analysis: task complexity, verboseness, number of expressions, number of Xpath expressions, Xpath length. In addition, the following user related predictors were included: hours of XSLT experience, hours of XQuery experience, number of participated courses, average score on all participated courses, number of participated programming courses, average score on all participated programming language courses, and general experience of the subjects. A description of these predictors can be found in Section 6.3.4, "Design and Analysis". We show the results of the regression analyses in the following sections. This regression analysis is carried out twice: for XSLT performance and separately for XQuery performance.

6.4.4.1. Correctness

For XSLT, we found that approximately 46% of the variance of correctness (Multiple $R^2 = .456$, $F(5,56)=9.39$, $p < .0005$) was explained with predictors that are presented in the table below. For XQuery, we found that approximately 22% of the variance of correctness ($R^2 = .223$, $F(2,59)=8.17$, $p < .001$) was explained with the same predictors that are presented below. The predictors are ordered by their beta values, that is, for each language the predictors are ranked according to their importance of contribution to the model. It is important to realize that the regression outcomes only indicate how much of the differences in performance within a language can be explained by the predictors.

Table 6.5. Predictors for Correctness

XSLT (Multiple R^2 = .456)	**XQuery (Multiple R^2 = .223)**
Number of programming courses (Beta = .752, sig = .000)	Task complexity (Beta = -.346, sig = .004)
Task complexity (Beta = -.574, sig = .000)	Number of programming courses (Beta = .321, sig = .008)
Hours of XQuery experience (Beta = .405, sig = .022)	
Average score on programming courses (Beta = .374, sig = .001)	
Xpath length (Beta = .314, sig = .048)	

This shows that both task complexity and number of programming courses of the subjects are the best predictors for correctness with XSLT , and the only predictors for correctness with XQuery. For both languages we found that the correctness on query tasks becomes higher when the number of programming language courses is higher and the complexity is lower. Other predictors for correctness with XSLT are the hours of XQuery experience, the average score on programming courses, and the length of Xpath expressions. When the values of these predictors are higher, the correctness becomes higher.

The following language related predictors were not significant for XSLT and XQuery: verboseness, number of expressions, and number of Xpath expressions. In addition, Xpath length was not significant for XQuery. User related predictors that were not significant for both XSLT and XQuery included hours of XSLT experience, number of participated courses, average score on all participated courses, and general experience of the subjects. In addition, hours of XQuery experience, and the average score on all participated programming language courses were not significant for XQuery. These results are discussed in more detail in Section 6.5, “Discussion”

6.4.4.2. Time

Subjects can solve query tasks incorrectly both in a very short and in a very long time. Therefore, the time that is spent on a query task is only meaningful when the query task is solved correctly. For this reason, only query tasks were included in the analysis that were solved correctly by at least one subject (the mean correctness on that assignment should be higher than zero). Note that we also analyzed the data with all query tasks, but this did not lead to better predictive models (lower R^2 scores). In this analysis, the same predictors as with the analysis of correctness were included.

For XSLT, we found that approximately 41% of the variance of time (Multiple R^2 = .411, F(3,53)=12.35, p < .0005) was explained with predictors that are presented in the table below. For XQuery, we found that approximately 31% of the variance of time (Multiple R^2 = .313, F(1,55)=25.02, p < .0005) was explained with the predictors that are presented below.

Table 6.6. Predictors for Time

XSLT (Multiple R^2 = .411)	**XQuery (Multiple R^2 = ..313)**
Xpath length (Beta = .334 sig = .024)	Verboseness (Beta = .559, sig = .000)
Verboseness (Beta = .316 sig = .031)	
Number of programming courses (Beta = -.325, sig = .004)	

For the time spent on query tasks with a correctness rate greater than zero, is verboseness for both XSLT and XQuery a predictor. The lengthier the query, the more time the subjects spent on the query tasks. For XSLT, we also found that Xpath length and number of programming courses were predictors for the time that subjects needed to complete a query. Longer Xpath expressions in the XSLT queries resulted in more time, and subjects who attended a larger number of programming courses needed less time to solve a query with XSLT.

The predictors that were included in the regression model, but were not significant are task complexity, number of expressions, number of Xpath expressions, hours of XSLT experience, hours of XQuery experience, number of participated courses, average score on all participated courses, number of participated programming courses, average score on all participated programming language courses, and general experience of the subjects. In addition, Xpath length and number of programming courses were no significant predictors for XQuery time. These results are discussed in more detail in the following section.

6.5. Discussion

The subjects in the XSLT group were in some aspects more experienced than the subjects in the XQuery group. We included the measurements of subject experiences as covariates in an Ancova, and found that there are no significant differences for the correctness of queries between XSLT and XQuery. For time nearly significant differences were found between XSLT and XQuery, where XQuery tasks were performed in less time compared to XSLT tasks. We found that the satisfaction of XQuery was nearly significant higher compared to XSLT.

XML document type did not appear as a predictor for performance as shown in Table H.3, “Predictors for correctness” and Table H.4, “Predictors for time”. There

are two possible reasons for this. First, it is possible that there is no influence of document structure on the performance with XSLT and XQuery. Second, the XML documents used in the experiment may have not reflected the differences between text-oriented and data-oriented well enough. Although we cannot completely exclude the first reason, we think that the document *size* is the most important reason why data structure did not appear as a relevant factor. Both types of documents were fairly small, so the time for the subjects to understand the documents was minimized. The differences between the document-oriented and the data-oriented document type were probably too small to make a difference in performance compared to factors like task complexity. To obtain more consistent and reliable data for each query task, we combined the same query tasks that were defined for the two document types to one query task.

For the most important question of this experiment, the results can be summarized as follows. For XSLT, 46% of the variance in correctness and 41% of the time spent on the query tasks was explained. The most important predictors for correctness with XSLT were task complexity and number of programming courses. The higher the task complexity, the lower the correctness. Correctness increased when the subjects had attended more programming courses. We also noticed that the correctness of XSLT tasks increased when the subjects who performed the tasks had more hours of XQuery experience and when the mean grades for programming courses were higher. Finally, the correctness increased when the Xpath expressions were longer. The predictors for the time of completing correctly solved XSLT queries were Xpath length, verboseness, and number of programming courses. The higher the number of programming courses, the faster the queries were solved and when the length of queries and Xpath expressions increased the slower the queries were solved.

For XQuery, 22% of the variance in correctness and 31% of the variance in time was explained. Correctness of the task was explained by the task complexity and the number of programming courses, where complexity was a slightly more important predictor. The more complex the task, the lower the correctness rate. The correctness rate for a query was higher when the number of programming courses was higher. Query length was the only significant predictor for time. The lengthier the query, the lower the performance and the more time it takes to solve a query.

We conclude that for correctness of the query task, task complexity and number of programming courses are for both XQuery and XSLT the most important predictors. For XSLT is programming experience more important than task complexity. The following table shows the mean number of programming courses per query task for both XSLT and XQuery:

Table 6.7. Number of Programming Courses

Language	Mean (Stdev.)	N
XSLT	5.9 (1.8)	61
XQuery	4.7 (3.0)	61

An independent samples t-test was used to compare these mean values, and the difference was significant: $t(60) = 2.30$, $p = .025$. Therefore, the number of programming courses of the subjects per XSLT query task is higher compared to the number of programming courses of the subjects per XQuery query task. Interestingly, no significant different mean values for programming experience were found between the total *groups* of XSLT and XQuery subjects. See also in Section 6.4.1, "Prior Experience of the Subjects". The reason for this is that the subjects with less programming experience attempted and solved less queries compared to the subjects with more programming experience. This means that to reach similar correctness rates on a set of tasks, more programming experience is required for XSLT compared to XQuery. In addition, correctness of XSLT tasks is predicted by the average grades on programming courses, hours of XQuery experience, and the length of Xpath expressions. The hours of XQuery experience was a significant predictor for the correctness rates with XSLT. An explanation for the significance of this predictor, is that the subjects with more hours of XQuery experience in the XSLT group are also the subjects who participated in the previous experiment. Therefore, these subjects may have had more recent experience with XSLT compared to the subjects who did not participate in the previous experiment. Subjects who participated in the previous experiment may also have been more accustomed to working with XSLT as a query language.

At first sight, the positive beta value for Xpath length as predictor for XSLT correctness is rather surprising. However, this only means that lengthier Xpath expressions predict a higher correctness on the query task when the influence of the other predictors is removed. In this case, we also noticed that task complexity was the most important predictor for correctness and that there was a positive significant correlation between Xpath length and task complexity ($r = .766$, $n=63$, $p < .0005$, one tailed.) Therefore, the correctness on query tasks increases with lengthier Xpath expressions when the influence on correctness of task complexity is removed. We think that longer Xpath expressions may be easier to formulate in XSLT when the complexity remains the same, for instance because when longer Xpath expressions are used there is no need to use XSLT constructs. In short, Xpath expressions are easier to formulate than XSLT expressions and longer Xpath expressions result in less XSLT syntax.

Time is predicted by verboseness for both languages. Apparently, the formulation time of a query is for both languages dependent on the number of characters that need to be written. See the following table for the mean values of verboseness for query tasks that are solved correctly at least once for both XQuery and XSLT:

Table 6.8. Query Length

Language	Mean	(Stdev.)	N
XSLT	300	(54)	53
XQuery	82	(41)	53

An independent samples t-test was used to compare the mean values for tasks that were solved correctly both for XSLT and XQuery. The difference in verboseness between XSLT and XQuery is significant: t(52) = 666,786, p = .000. This means that the difference in time between XSLT and XQuery can partially be explained by the difference in verboseness. In addition, for XSLT also Xpath length and number of programming courses are relevant predictors. When subjects had followed more programming courses, they wrote XSLT queries faster than subjects who had attended less programming courses. More lengthy Xpath expressions resulted in more time to formulate XSLT queries.

The number of expressions was not found as a significant predictor in the regression models. We included this predictor because we expected that the number of expressions would be smaller when specialized operators could be used. However, this relation is probably too weak in reality. The conclusions of this experiment are discussed in the following section.

6.6. Conclusions

We conclude with the results of this experiment and the answers to the research questions that we formulated in Section 6.2, “Research Questions”.

Performance and Satisfaction. Research question 1 and 2 were related to the performance with, and satisfaction of, XSLT and XQuery. We have shown that performance of XQuery on a large set of query tasks is better than performance with XSLT when we included the subject's experience as covariates in the analysis. Although there are no differences for the correctness rates, the time to solve a query is shorter for XQuery compared to XSLT. Therefore, XQuery is more efficient to use. XQuery also leads to higher satisfaction than XSLT.

Predicting performance. The third research question was related to predictors of performance with XSLT and XQuery. We were able to predict 46% of the correctness and 41% of the time spent on queries with XSLT. For XQuery, we were able to predict 22% of the correctness and 31% of the time spent on queries. The results regarding the predictors we used are summarized below.

Complexity of the Query Task. This is for XQuery the most important predictor for correctness and for XSLT it is an important predictor for the correctness. The more complex the task, the lower the correctness rates with XSLT and XQuery.

Experience of the User. Programming experience is an important predictor for XSLT performance, both correctness and time. For XQuery it is a relevant factor to predict correctness of the result. The more experienced the user, the better the performance with XSLT and the higher the correctness with XQuery. We noticed that XSLT queries were solved by subjects who had more programming experience compared to the subjects who used XQuery. The overall correctness rates were the same for XSLT and XQuery, so this means that for the same correctness rates on the same set of tasks, more programming experience is needed for XSLT. This indicates that XSLT is harder to use and learn and this can, at least partially, be explained with the performance predictors that are strictly query language related: verboseness and Xpath embedding.

Verboseness of the Language. Verboseness is represented by the number of characters that was needed to formulate a query. Verboseness was a predictor for time for both XSLT and XQuery. For both languages applies, the more characters are needed to formulate a query, the more time is needed. We have also seen that XSLT expressions consist of more characters than XQuery expressions. Therefore, we conclude that one of the reasons why it took longer to formulate XSLT expressions compared to XQuery expressions is the verboseness of XSLT.

Embedding of Xpath Expressions. Two predictors were used to represent Xpath embedding: Xpath length and number of Xpath expressions. The number of Xpath expressions did not appear as a predictor in the regression models of XSLT and XQuery performance. However, Xpath length was a predictor for XSLT correctness and time. The correctness rates with XSLT became higher when longer Xpath expressions were used and it took longer to solve queries. Xpath length was no predictor for XQuery performance. This shows that Xpath embedding is more difficult to use in XSLT compared to XQuery.

Number of Expressions. The number of expressions was no predictor for performance with XQuery or XSLT.

XML Document Structure. Document structure was no predictor for performance with XQuery and XSLT, see also the result when this factor was included in Appendix H, *Results for Assignments*

Chapter 7. Conclusions

7.1. Introduction

In this chapter we present a summary of the results and discuss the contributions of this study. We conclude with directions for further research.

7.2. Summary of the Results

Chapter 2 - Background. In this chapter we described the ideas and findings prior to this study. There are two main lines of discussion in this chapter. First, the opinions in the XML community about the functionality of XML query languages. Second, the findings of research to the ease of use of database query languages. In the XML community there are ongoing discussions about which language is easier to learn and use and for which application. Common arguments are that XSLT is more difficult to learn and use due to two reasons. First, the *verbose* XML syntax. Second, the functional nature of XSLT which becomes most apparent in the use of *template rules*. However, these template rules make XSLT probably more suitable for document-oriented XML applications. XQuery on the other hand, has a SQL-like and more compact syntax which is claimed to be easier to understand. XQuery is claimed to be more suitable for data-oriented applications, but the arguments for this are mainly related to implementation issues that are outside the scope of this study. This overview is concluded with a brief description of three potential important XML query languages: XSLT, XQuery, and SQL/XML. From previous research to the ease of use of database query languages we learnt the following:

- Differences in user performance were found for different query languages on the same set of tasks. One research paper indicated that the level of procedurality influences the performance with a language, where procedurality is defined in terms of the numbers of expressions and variable bindings and their permissible ordering in a query.
- User performance was also influenced by user experience. Users who were more experienced programmers generally performed better with a new to learn query language.
- Differences in user performance were found for different types of queries. Performance with more complex queries like joins and grouping was generally lower than with less complex queries like a simple mapping query.
- The influence of the content and structure of a database on the performance with query languages has not been investigated properly yet. Some indications of possible influence of these aspects were found in previous studies.

- Two models of the human process of query solving were presented. A common idea of these models is to distinguish between the general query problem that needs to be solved and the syntactical constructions that are used to represent this problem. Other relevant features included the distinction between error types, and the use of a model to predict query performance on different types of queries.

Chapter 3 - Understanding the Query Process. In this chapter a model of the human process of query solving is presented. This model is based on models and error types that are described in chapter 2 and on our personal experience. The model is a detailed description of how users formulate query language expressions. According to this model, the process of query consists of several activities and subactivities. These activities include physical behavior like reading a reference guide and thinking or reasoning steps. The top level activities of the model include understand the instructions, construct the deep structure of a query, construct the surface structure of a query, submit the query, evaluate the result, and perform correction efforts. The performance differences between query languages noticed or assumed in the previous chapter must be related to how queries are solved with different query languages. Therefore, different query languages and different query language aspects can be explained in terms of the model of the query solving process. The ideas, terms, and concepts from the previous chapter are reformulated in research questions. We conclude that too little is known about this query language usability to design a quantitative experiment. First we need to gather more qualitative data about the human process of XML query formulation, which is performed in the following chapter.

Chapter 4 - Experiment 1: Exploring the Model. This chapter describes a thinking aloud experiment in which we tried to find indications for the correctness of the assumptions mentioned in chapter 2 and our understanding of the query solving process described in chapter 3. Another goal of this experiment is to find other potential factors that influence the usability of an XML query language. Most important findings of this experiment are:

1. Performance differences were found between the languages: XQuery performance was highest, XSLT intermediate, and SQL/XML performance was lowest.
2. A clear indication was found for the influence of query type: performance was higher on less complex queries and the differences between languages became most apparent with more complex queries.
3. The behavior of querying subjects could be explained with the model we developed. The effectiveness of the users was related to the number of activities during the query solving process, most notably in the activities related to the construction of the surface structure, as mentioned in the thinking aloud protocols.
4. We found indications that the following factors can influence the user performance during the query solving process: consistency, verboseness, expression type, and Xpath embedding. We found no support for the influence of procedurality of a language.

5. XSLT users made more Xpath related errors compared to both SQL/XML and XQuery, probably due to the way Xpath is embedded in XSLT.
6. SQL/XML was experienced as a very unsatisfying language by subjects in the experiment. Subjects were frustrated by the use of two data models in one language and the frequently required use of compound expressions for very simple query tasks. XQuery and XSLT were appreciated equally positive.

The model we used to explain the behavior of subjects in the experiment contained more detailed information than was required to answer the research questions. Also, we found no support in the data for a few processes and transitions in the model. Therefore, we presented a simplified model of the query solving process based on the findings in this experiment.

Chapter 5 - Experiment 2: Validation. In this chapter we tried to validate our findings of the previous chapter with a quantitative experiment with a group of 74 bachelor students Information Science who participated in an introductory XML course. SQL/XML was excluded from this experiment because subjects in the thinking aloud experiment were frustrated by this language and because SQL/XML was less suitable to include in the curriculum of the students due to the preliminary status and implementation of this language. We showed that the usability (performance and satisfaction) of XQuery was higher than XSLT on five representative query tasks, despite the significantly shorter preparation time with XQuery. In contrast to the results of the thinking aloud experiment, we did not find that the performance differences between XSLT and XQuery became larger with more complex queries. We concluded that this is due to the different experience levels of the subjects of the first and the second experiment. Also, the queries in this experiment may have been, unintentionally, more complex than the queries in the previous experiment. In addition, we concluded that the common binary distinction between less complex and more complex queries was too coarse to be useful in this experiment. In this experiment we also tested hypotheses related to specific query task and query language related aspects. We found strong indications for the influence of the following factors:

1. Verboseness: a more verbose language results in lower performance;
2. Xpath embedding: performance with XSLT on queries with a larger number of Xpath expressions was lower compared to performance with XQuery;
3. Query type: we noticed that performance on more complex queries was generally lower compared to the performance on less complex tasks;
4. Expression type: we noticed that a specialized grouping expression in XSLT resulted in better performance compared to the use of compound expressions in XQuery for the same query task. However, on another query the use of a specialized XQuery expressions did not led to better performance compared to the use of a compound expression in XSLT.

In addition, we also noticed a strong influence of user experience. A third XML query language usability experiment was conducted to explain the reasons for performance

differences between XQuery and XSLT in more detail, and to make a first attempt to investigate the influence of document structure on the query solving process.

Chapter 6 - Experiment 3: Explanation. In this chapter we explained the differences in usability between XQuery and XSLT with a third experiment. Instead of using five representative query tasks, we selected 64 query tasks from various sources. We developed and applied a method to determine a scalar representation of the complexity of query tasks instead of distinguishing between less complex and more complex queries. In our previous experiment we noticed that enforcing a particular query strategy to the subjects led to unexpected results, so this method was not used in this experiment. Therefore, we were not able to enforce the use of a particular expression type in this experiment. Instead, we counted the number of expressions because a specialized operator normally results in less (compound) expressions. We compared the performance on a large number of different tasks (128) where each task is performed by a few subjects (2-6). The reason for this, is that we assume that the natural variance of several factors measured over a large set of queries enables us to predict usability with XQuery and XSLT. The following factors were included:

1. Experience of the users that attempted a particular task. User experience is expressed in several aspects, like hours of XSLT and XQuery experience, number of attended programming courses, and academic achievements.
2. Document type. Two different XML document structures were used, a data-oriented XML document and a document-oriented XML document.
3. Verboseness of the query language. Verboseness is represented by the number of characters of query expressions.
4. The number of expressions, i.e. XSLT elements and XQuery keywords, that are used to solve a particular query task.
5. Xpath embedding in the query language. This factor is represented by the number of Xpath expressions and the number of characters of Xpath expressions in a query language expression.

The results show that performance and satisfaction are higher with XQuery compared to XSLT when we included measurements for the subject's experience as covariates. The factors mentioned above were shown to influence the query process, except for the factors document structure and number of expressions. The other factors influenced both languages but not always in the same way. For instance, programming experience was more important for XSLT than for XQuery. The reason why we found no indications for the influence of XML document structure was probably due to the relative easy structure and small size of both the data-oriented and the document-oriented XML documents.

7.3. Contributions of this Study

In chapter 1 we formulated the following global research questions:

1. What is the usability of most important XML Query Language candidates?
2. What are the reasons for differences in usability between these languages, if any? In particular, what is the influence of the following factors:
   a. Syntax features of the language;
   b. Background of the users;
   c. Structure of the XML documents.

Contributions of this study that were not expressed in advance in the research questions are summarized in:

3. Other contributions:
   a. Model of the Query Solving Process;
   b. Query Complexity;
   c. Methodology.

First, we provide the answers to the research questions.

1. Usability of XML Query Languages. Our main conclusion is that the usability of XQuery is higher compared to XSLT. The results on three experiments show that either or both the effectiveness and efficiency of subjects are higher compared to XSLT. Subjects were more satisfied with XQuery compared to XSLT. The exploration of SQL/XML in the first experiment made us decide that the usability of SQL/XML in the current stage of development was too low to include in the remainder of the study. We found multiple reasons for usability differences between the languages and several factors that influence the query solving process. We state them below.

2a. Syntax Features. The following syntax aspects were considered in this study:

- *Consistency:* based on the comparison of SQL/XML with XSLT and XQuery in the first experiment, it is highly likely that inconsistencies of a language have a negative influence on its usability. Subjects performed lower due to inconsistent names of functions and considered these inconsistencies as a weak aspect of the language.
- *Verboseness:* in this study we found that the verboseness of a language has a negative effect on the language's usability. In the first experiment we noticed that performance with the verbose language XSLT was lower compared to the more compact XQuery. Subjects in this experiment explicitly stated that XSLT's verbose XML syntax was not a good choice for a query language. This result was confirmed in the second and third experiment. In experiment 2 we noticed that the performance with XSLT was lower on a task where verboseness was the main difference between

the XQuery and XSLT solutions. In the third experiment we clearly demonstrated that when more characters are needed to formulate a query in both XSLT and XQuery, the longer it takes to solve the query. XSLT generally results in longer queries compared to XQuery and therefore the verboseness of XSLT has a negative influence on the efficiency with this language.

- *Xpath embedding:* the method of Xpath embedding in XSLT has a negative effect on the usability of this language compared to XQuery. In the first experiment we noticed that more Xpath related errors were made with XSLT compared to both SQL/XML and XQuery. In the second experiment we found that performance on queries that included a large number of Xpath expressions was lower with XSLT than with XQuery. This result was further explained in the third experiment, where we found that the length of Xpath expressions only in XSLT predicted both the correctness of queries and the time spent on queries.
- *Expression type:* In the first and second experiment we noticed that XSLT performance was better on grouping queries. This is due to the for-each-group element which we classified as a specialized expression. However, in the second experiment we found no difference between XSLT and XQuery when a specialized expression was used in XQuery and not in XSLT. In the third experiment we formulated the idea that a general effect of the use of specialized expressions would be that the number of expressions would become smaller. However, we found no support for this idea. Therefore, the question remains: what is the effect of a specialized expression on user performance? In chapter 6 we stated that counting the number of expressions is probably not a good method to represent specialized expressions. In the second experiment we used two query tasks to test the difference between compound and specialized expressions: a query that selects unique nodes from the XML document and a grouping query. These queries have different deep structures and are not equally complex. We noticed that the specialized expression resulted in better performance on the complex query only. Furthermore, when we looked more closely to the query results on the less complex queries, we had the impression that the subjects using XQuery misinterpreted the query task description while this was not the case with the subjects using XSLT. Therefore, the lack of a performance difference may have other reasons. We argue that specialized expressions are in particular useful for learning how to solve more complex problems. In other words, a specialized expression will make it easier for users to apply an appropriate surface structure to a deep structure of a complex query.

2b. Background of the Users. In this study we demonstrated that the user's background has a large influence on the performance with XML query languages. Informally, we noticed this when comparing the performance of subjects in the first and second experiment. The subjects who participated in the first experiment were database researchers and were more experienced than the subjects in the second experiment who were first year Information Science students. The less complex queries were easily solved by the subjects in the first experiment, whereas in the second experiment some subjects had serious problems with this type of queries. Although

the queries in the second experiment may have been slightly more complex, it is very clear that the more experienced subjects in the first experiment performed better compared to the subjects in the second experiment. In the third experiment we showed that to achieve similar correctness rates with XSLT, more programming experience is needed than with XQuery. Note that the results of this study are tested with users who learned and practiced with the query languages in a short period of time. The results of this study must therefore be interpreted as representative for the beginning of the learning curve of the XML query languages.

2c. Structure of XML Documents. A goal of this study was to investigate the influence of different XML structures on the performance with XML query languages. In the first experiment we informally noticed some minor problems with counter intuitive structures. In the third experiment we tested whether document-oriented and data-oriented XML documents resulted in performance differences. No user performance differences were found with XSLT and XQuery between these XML documents. However, the documents used in this experiment may have been too small to represent the differences between the application types.

In addition to answering the research questions, the following valuable contributions are made through this study.

3a. Query Complexity. Based on earlier studies and on the results of the first experiment we expected that performance differences between query languages would become larger on more complex query tasks than on less complex query tasks. When we tried to validate this idea in the second experiment, we noticed that: a) level of task complexity may be relative to the user experience and b) the binary distinction between less and more complex was too general to classify the queries in the experiment. Therefore, we developed a scalar measurement for query task complexity. We were able to predict user performance with this complexity measurement. In experiment 3, we showed that complexity is an important performance predictor for both XSLT and XQuery. The more complex the query, the lower the performance. Although this is rather obvious, the measurement for task complexity may be useful in other contexts too.

3b. Model of the Query Solving Process. A model of the query solving process is proposed in this study. We explored this model in a thinking aloud experiment, and the behavior of querying subjects could be explained in terms of this model. In addition, the influence of specific language aspects was related to specific subactivities of the model. This model may be used to model user behavior with other query languages and perhaps even with certain programming languages as well.

3c. Methodology. In this study we demonstrated that usability evaluation cannot only be applied successfully to, for instance, graphical user interfaces but also to XML query languages. The general ideas and methods can also be used to evaluate other

query or programming languages. The methods deployed in this study can therefore inspire other researchers to apply these methods to other research topics.

7.4. Directions for further Research

XML Query Language Aspects. During the design phase of a new formal language it can also be useful to evaluate the usability of language alternatives. A thinking aloud experiment like we conducted in chapter 4 can be executed with a limited set of subjects but can provide very useful information and new insights. For instance, our results show that the usability of XQuery would increase when a grouping operator is included in the language. The usability of XSLT would increase when the verboseness could be reduced and the method of Xpath embedding could be simplified. In this study, the basic subset of the query languages were used by the subjects. Support for, for instance, XML Schema Data types and the use of quantifiers were not included in this experiment because of limitations of the language processors and the relative short learning time of the subjects. Also, we found no positive effect of procedurality on the user performance. However, the non-procedural nature of XSLT may have remained implicit because subjects formulated their XSLT queries with only one or two templates. This study investigated the start of the learning curve of three XML query languages. In the future, it would be very interesting to focus on more advanced features of these languages and on more complicated query tasks. This would also require more experienced users of XQuery and XSLT.

XML Document Structure. In this study we did not find support for the idea that data-oriented and document-oriented XML structures result in performance differences. However, we suspect that the documents included in this study were not representative enough. Therefore, future research is required to gain a better understanding of the effect of XML document structure on the performance with XML Query languages.

Usability Benchmarks. Several benchmarks are available to test machine performance of XML query processors or search engines. However, these collections are not very suitable to test usability aspects of XML querying. In this study, we constructed a large number of query tasks and documents that were used to determine the user performance with XML query languages. Some of these tasks were very suitable, while others were more problematic. For instance, the directed query tasks in the second experiment added unexpected and unintended characteristics to the query tasks. We noticed that less experienced user performed better with one "fill in the blanks" query, while more experienced users performed worse due to the provided template of the query. The effect of providing a template of a query is not completely understood. During this study, we included different query tasks and different scales to score the result queries. It would be very rewarding to construct usability benchmarks with fixed sets of XML documents, query tasks, and scoring systems to rate the result of a query. The set of tasks constructed for the third experiment may function as a first attempt to such a benchmark.

Bibliography

[ABC+01] C. Arms, M. Banks, P. Case, et al. XML Query Use Cases from the library of congress. http://www.loc.gov/crsinfo/xml/lc_usecases.html, 31 May 2001.

[BB82] K. F. Bury, J. M. Boyle. An on-line experimental comparison of two simulated record selection languages. Proceedings of the Human Factors Society 26th annual meeting, pages 74-78, Santa Monica CA, 1982.

[BBE83] J. M. Boyle, K. Bury, R. J. Evey. Two studies evaluating learning and use of QBE and SQL. Proceedings of the Human Factors Society 27th Annual Meeting, pages 663-667, Santa Monica CA, 1983.

[BCM+96] A. N. Badre, T. Catarci, A. Massari, G. Santucci. Comparative ease of use of a diagrammic vs. an iconic query language. Interfaces to databases. Electronic Series in Computing. pages 1-14, 1996.

[BLL+01] S. Bressan, M. L. Lee, Y. G. Li, Z. Lacroix, U. Nambiar. The XOO7 XML Management System Benchmark. NUS CS Department Technical Report TR21/00. November, 2001.

[BR00] M. Boren, J. Ramey. Thinking aloud: Reconciling Theory and Practice. IEEE Transactions on Professional Communication. 43 (3), pages 261-278, 2000.

[BS78] M. Brosey, B. Shneiderman. Two experimental comparisons of relational and hierarchical database models. International Journal of Man-machine studies. 10, pages 625-637, 1978.

[BS96] L. Bainbridge, P. Sanderson. Verbal Protocol Analysis. Evaluation of Human Work. 169-201, J. R. Wilson, E. N. Corlett, Taylor and Francis, London, 1996.

[CRF00] J. Robie, D. Chamberlin, D. Florescu, Quilt: An XML Query Language for Heterogeneous Data Sources. http://www.almaden.ibm.com/cs/people/chamberlin/quilt_euro.html, 31 March 2000.

[CS95] T. Catarci, G. Santucci. Are visual query languages easier to use than traditional ones? An experimental proof. International conference on Human-Computer Interaction (HCI95), 1995.

[Cuf80] Cuff, R. N. On casual users. International Journal of Man-Machine Studies, 12, pages 163-187, 1980.

[DFF+99] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, D. Suciu. A Query Language for XML. pages 1155-1169, 31, Computer Networks, 1999.

[FHH00] E. Frøkjær, M. Hertzum, K. Hornbæk. Measuring Usability. CHI Letters, 2(1), pages 345-352, 2000.

[GGD86] S. L. Greene, L. L. Gomez, S. J. Devlin. A cognitive analysis of database query production. Proceedings of the Human Factors Society 30th annual meeting, pages 9-13, Santa Monica CA, 1986.

[Gra03] J. Graaumans, Neglected Requirements for XML Query Languages. Interchange, 9 (2), pages 34-38, 2003.

[Gra04] J. Graaumans. A qualitative study to the usability of three XML Query languages. SIGCHI, Amsterdam, Netherlands, 2004.

[Gra05] J. Graaumans. A Collection of XML Documents and Query Tasks. Technical report, Utrecht University, UU-CS-2005-038, 2005.

[Gu04] Gupta, D. What is a good first programming language? ACM Crossroads Computer Science Education, 10. 4, Summer 2004.

[GW78] Greenblatt, D. J. Waxman, A study of three database query languages. B. Shneidermann. Databases: improving usability and responsiveness, New York, Academic Press. 1978.

[JV85] M. Jarke, Y. Vassiliou. A framework for choosing a database query language. Computer Surveys, vol. 17, no. 3, September 1985.

[Kay04] M. Kay. XQuery, Xpath, and XSLT. XQuery from the experts: A Guide to the W3C XML Query Language, H. Katz. Addision-Wesley, 2004. ISBN: 0-321-18060-7.

[Kay98] M. Kay. XSLT: programmers reference. Wrox Press, May 15, 2000. ISBN: 1861003129.

[LN77] P. H. Lindsay, D. A. Norman. Human Information Processing: An Introduction to Psychology. Academic Press, New York, 1977.

[Loc78] F. H. Lochovsky, Data Base Management System User Performance. Ph.D. Thesis, Department of Computer Science, University of Toronto, 1978.

[LT77] F. H. Lochovsky, D. C. Tsichrizis. User performance considerations in DBMS selection. Proceedings of ACM SIGMOD, pages 128-134, 1977.

[Nie93] J. Nielsen. Usability Engineering. Morgan Kaufman, 1993. ISBN: 0-12-518406-9.

[QL'98] M. Marchiori. Homepage of QL'98 - Query Languages 1998. http://www.w3.org/TandS/QL/QL98/, December 5, 1998.

[Rau96] M. Reuterberg. How to measure cognitive complexity in Human-Computer Interaction? Proceedings of the Thirteenth European Meeting on Cybernetics and Systems Research, pages 815-820, Austrian Society for Cybernetics Studies, 9-12 april 1996.

[RBC75] P. Reisner, R. F. Boyce, D. Chamberin. Human factors evaluation of two database query languages - Square and Sequel. Proceedings of the

National Computer Conference. pages 447-452, Arlington VA: AFIPS Press. 1975.

[Rei77] P. Reisner. Use of psychological experimentation as an aid to development of a query language. IEEE TRansactions on software engineering, SE-3, pages 218-229, 1977.

[Rei88] P. Reisner. Query languages. Handbook of Human Computer Interaction. M. Helander. Elsevier science publishers, 1988. ISBN: 0 444 88673 7.

[Robie04] J. Robie. XQuery: A Guided Tour. XQuery from the experts: A Guide to the W3C XML Query Language, H. Katz. Addision-Wesley, 2004. ISBN: 0-321-18060-7.

[RPJ+03] K. Runapongsa, J. M. Patel, H. V. Jagadish, Y. Chen, S. Al-Khalifa. The Michigan Benchmark: Towards XML Query Performance Diagnostics. Proceedings of the International Conference on Very Large Data Bases (VLDB), Berlin, Germany, 2003.

[Saxon] M. Kay. SAXON: The XSLT and XQuery Processor. http://saxon.sourceforge.net/.

[Schl91] M. Schlager. An analysis of database querying as a cognitive skill. Ph.D. Thesis, Department of Psychology, University of Colorado, 1991.

[Smi01] M. Smith. XQuery, XSLT "overlap" debated. http://www.xmlhack.com/read.php?item=1080, 23 Feb 2001.

[SQL/XML] J. Melton. (ISO-ANSI Working Draft) XML-Related Specifications (SQL/XML). http://sqlx. org/5wd-14-xml-2002-08.pdf, August, 2002.

[SWK+02] A. R. Schmidt, F. Waas, M. L. Kersten, M. J. Carey, I. Manolescu, R. Busse. XMark: A Benchmark for XML Data Management. Proceedings of the International Conference on Very Large Data Bases (VLDB), Hong Kong, China, pages 974-985, August 2002.

[TG75] J. C. Thomas, J. D. Gould. A psychological study of query by example. Proceedings of the National Computer Conference, pages 439-445, Arlington: AFIPS Press, 1975.

[VdS01] G. van der Steen, Naar de menselijke maat, Het perspectief van Uitwisselingstalen. 2001.

[VZw04] R. van Zwol, H. van Oostendorp. Google's 'I'm feeling lucky', Truly a Gamble? Proceedings of the Fifth International Conference on Web Information Systems Engineering, 2004.

[W3C-FTR] S. Buxton, M. Rys. XML Query and XPath Full-Text Requirements. W3C Technical Report, http://www.w3.org/TR/xquery-full-text-requirements/, 02 May 2003.

[W3C-FTU] S. Amer-Yahia, P. Case. XQuery 1. 0 and XPath 2. 0 Full-Text Use Cases. W3C Technical Report, http://www.w3.org/TR/xmlquery-full-text-use-cases/, 4 April 2005.

[W3C-QL] S. Boag, D. Chamberlin, M. F. Fernandez, D. Florescu, J. Robie, J. Simeon. XQuery 1. 0: An XML Query Language. W3C Technical Report, http://www.w3.org/TR/xquery/, 05-04-2005.

[W3C-QR] D. Chamberlin, P. Fankhauser, M. Marchiori, J. Robie. XML Query requirements. W3C Technical Report, http://www.w3.org/TR/xmlquery-req, 15-02-2001.

[W3C-QU] D. Chamberlin, P. Fankhauser, D. Florescu, M. Marchiori, J. Robie. W3C XML Query Use Cases. W3C Technical Report, http://www.w3.org/TR/xquery-use-cases/, 04 April 2005.

[W3C-S2] P. V. Biron, A. Malhotra. W3C XML Schema, part 2: Data types. W3C Technical Report, http://www.w3.org/TR/xmlschema-2/, 28 October 2004.

[W3-S] W3C School. http://www.w3schools.com/

[WS81] C. Welty, D. W. Stemple. A human factor comparison of a procedural and a nonprocedural query language. ACM TRansactions on Database Systems, 6, pages 626-649, 1981.

[XML] T. Bray. Extensible Markup Language (XML) 1. 0 (Second Edition). W3C Recommendation, http://www.w3.org/TR/REC-xml, 6 October 2000.

[XML-DEV] Monthly Archives for xml-dev. http://lists. xml. org/archives/xml-dev/.

[Xpath] J. Clark, S. DeRose. XML Path Language (XPath) Version 1.0. W3C Recommendation. http://www.w3.org/TR/xpath, 16 November 1999.

[XSL] The Extensible Stylesheet Language Family (XSL). http://www.w3.org/Style/XSL/.

[YOK02] B. B. Yao, M. T. Oszu, J. Keenlyside. XBench - A Family of Benchmarks for XML DBMSs. December 2002.

Appendix A. Additional figures

Figure A.1. Understand Conditions of Search Activities

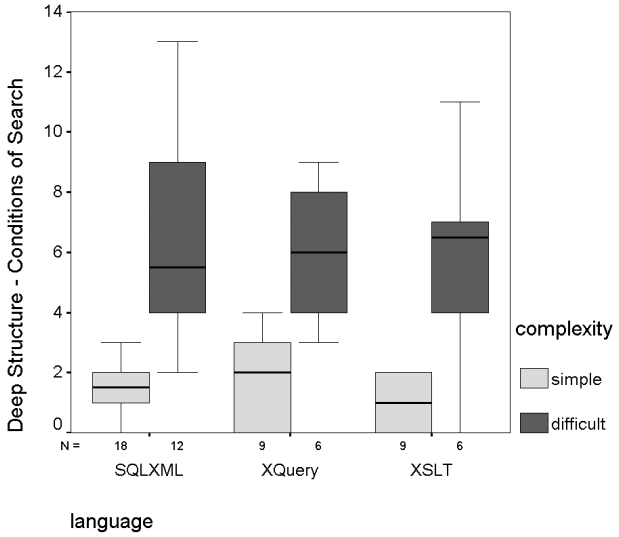


Figure A.2. Total Number of Activities - Join and Grouping

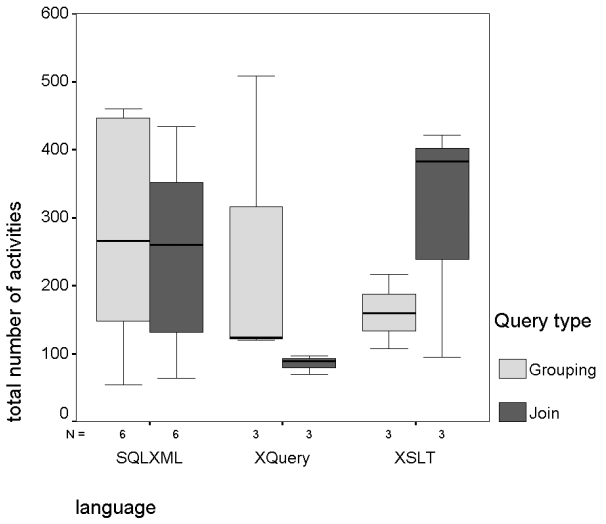


Figure A.3. From deep structure to surface structure

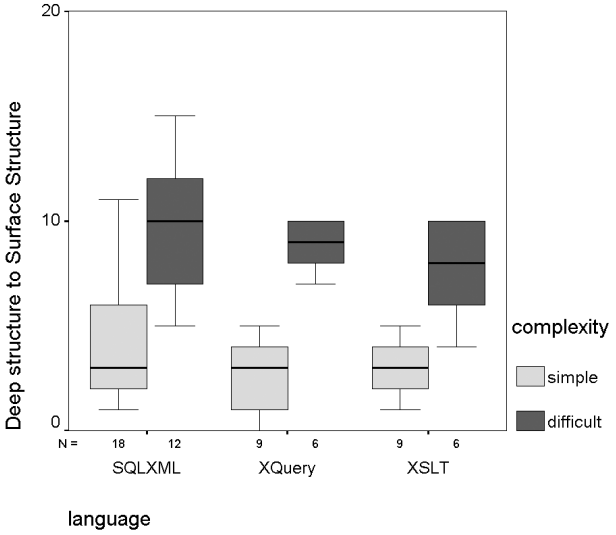


Figure A.4. From Surface Structure to Deep Structure

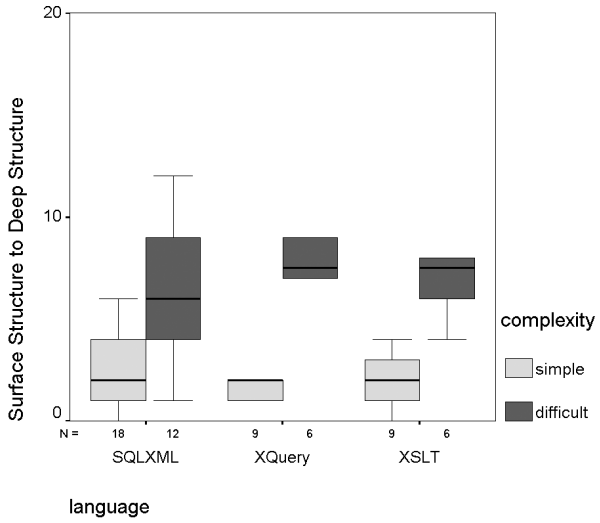
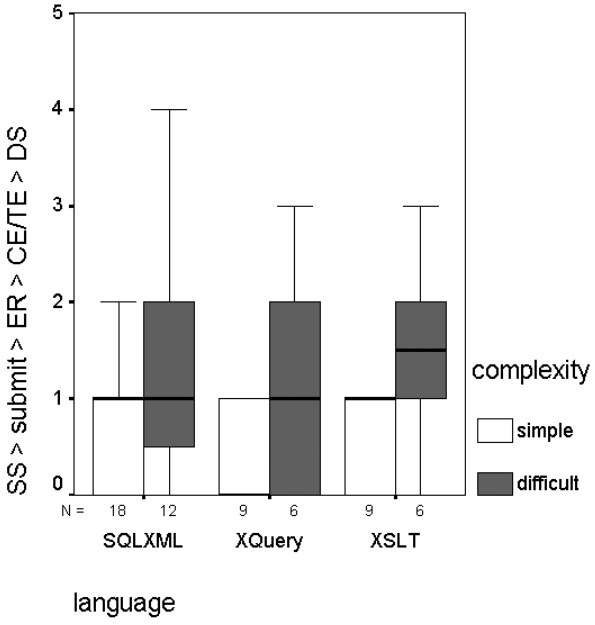


Figure A.5. Constructing and Debugging the Surface Structure



Appendix B. Questionnaire - Prior Experience

The following list contains the questionnaire we used in Chapter 5, *Experiment 2: Validation*. When a question includes the phrase XQuery/XSLT, this means that either XSLT or XQuery is used, depending on the language the subjects worked with.

1. What is your discipline?
2. What is your year of study?
3. Have you attended the course "Databases", or are you currently attending this course?
4. Have you successfully concluded the course "Databases"?
5. Have you attended the course "Imperative Programming", or are you currently attending this course?
6. Have you successfully concluded the course "Imperative Programming"?
7. Are you working (besides your studies) with XML, of have you worked with XML before?
8. Are you working (besides your studies) with HTML, of have you worked with HTML before?
9. Are you working (besides your studies) with XQuery/XSLT, of have you worked with XQuery/XSLT before?
10. Are you working (besides your studies) with SQL, of have you worked with SQL before?
11. Are you working (besides your studies) with functional programming languages?
12. Are you working (besides your studies) with other programming languages?
13. How much time did you spend studying the literature about XSLT?
14. Were you present at the lecture about Xpath?
15. Were you present at the first lecture about XSLT?
16. Were you present at the second lecture about XSLT?
17. Were you present at the lecture about XQuery?
18. How much time did you spend on the practical assignments of Xpath?
19. How much time did you spend on the practical assignments of XSLT?
20. How much time did you spend studying the literature about XQuery?
21. How much time did you spend exercising with XQuery?
22. It may be relevant for the outcomes of this experiment to consider your academic achievements. Can we have your permission to ask for your achievements at the faculty's office?

Appendix C. Questionnaire - Satisfaction

The following list contains the questionnaire we used in Chapter 5, *Experiment 2: Validation*. When a question includes the phrase XQuery/XSLT, this means that either XSLT or XQuery is used, depending on the language the subjects worked with.

1. I think that Xpath is a simple language.
2. I find it nice to work with Xpath.
3. Xpath expressions are simple and intuitive.
4. Longer Xpath expressions are more difficult to formulate.
5. Xpath is similar to a language I already know.
6. It is easy to use Xpath expressions in XQuery/XSLT queries.
7. I find XQuery/XSLT easy to learn.
8. I find it nice to work with XQuery/XSLT.
9. XQuery/XSLT is similar to a query language I already know.
10. It is easier to split Xpath expressions with XQuery/XSLT, than to formulate extensive Xpath expressions.
11. The syntax of XQuery/XSLT has a clear and simple structure.
12. My prior knowledge was sufficient to formulate the XQuery/XSLT queries.
13. My prior knowledge was sufficient to formulate the Xpath queries.
14. I found it easy to solve the XQuery/XSLT queries.
15. I found it easy to solve the Xpath queries.
16. I would like to use XQuery/XSLT more often in the future.

Appendix D. Experiment 2: XML Document

```
<?xml version="1.0" ?>
<authors>
  <author born="1802" died="1885">
    <name idd="VH">Victor Hugo</name>
    <nationality>French</nationality>
  </author>
  <author  born="-496" died="-406" period="classical">
    <name idd="SO">Sophocles</name>
    <nationality>Greek</nationality>
  </author>
  <author born="1828" died="1910">
    <name idd="LT">Leo Tolstoy</name>
    <nationality>Russian</nationality>
  </author>
  <author born="1799" died="1837">
    <name idd="AP">Alexander Pushkin</name>
    <nationality>Russian</nationality>
  </author>
   <author born="1615" died="1885">
    <name idd="VH">Jan Jaap van der Wal</name>
    <nationality>French</nationality>
  </author>
  <author  born="-427" died="-347" period="classical">
    <name idd="PL">Plato</name>
    <nationality>Greek</nationality>
  </author>
  <author  born="1885" died="1951">
    <name idd="SL">Sinclair Lewis</name>
    <nationality>American</nationality>
  </author>
  <author  born="1885" died="1940">
    <name idd="HW">Humbert Wolfe</name>
    <nationality>English</nationality>
  </author>
</authors>
```

Appendix E. Baseline Performance Tasks

To determine whether the subjects had a certain baseline performance level in experiment 2, five Xpath query tasks were used. These queries were the same for every subject. They are stated below:

1. Formulate an Xpath expression that selects all "nationality" elements.
2. Formulate an Xpath expression that selects all "name" elements with an "idd" attribute.
3. Formulate an Xpath expression that selects all "name" elements with an "idd" attribute with value "AP".
4. Formulate an Xpath expression that selects all "name" elements from "author" elements with a "nationality" with value "Russian".
5. Formulate an Xpath expression that selects all "author" elements with an attribute "died" with value "1885".

Appendix F. Document-oriented XML

F.1. Report.dtd

```
<!ELEMENT author (last, first)>
<!ELEMENT chapter (title, author+, intro?, section*)>
<!ATTLIST chapter
 number (1 | 2 | 3) #REQUIRED
>
<!ELEMENT emph (#PCDATA)>
<!ELEMENT first (#PCDATA)>
<!ELEMENT graphic EMPTY>
<!ATTLIST graphic
 graphname (infoflow | tagexamp) #REQUIRED
>
<!ELEMENT intro (para+, graphic?)>
<!ELEMENT keyword (#PCDATA)>
<!ELEMENT last (#PCDATA)>
<!ELEMENT para (#PCDATA | emph | xref)*>
<!ATTLIST para
 security (c | u) #IMPLIED
>
<!ELEMENT report (title, chapter+)>
<!ELEMENT section (title, intro, topic*)>
<!ATTLIST section
 shorttitle CDATA #IMPLIED
>
<!ELEMENT title (#PCDATA | emph)*>
<!ELEMENT topic (title, keyword?, para+, graphic?)>
<!ATTLIST topic
 id ID #IMPLIED
 number CDATA #IMPLIED
>
<!ELEMENT xref EMPTY>
<!ATTLIST xref
 idref IDREF #REQUIRED
>
```

F.2. Report.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE report SYSTEM "report.dtd">
<report>
  <title>Getting started with SGML</title>
  <chapter number="1">
    <title>The business challenge</title>
    <author>
      <last>Stevens</last>
      <first>W.</first>
    </author>
    <intro>
      <para>With the ever-changing and growing global market,
      companies and large organizations are searching for ways to
      become more viable and competitive. Downsizing and other
      cost-cutting measures demand more efficient use of corporate
      resources. One very important resource is an organization's
      information.</para>
      <para>As part of the move toward integrated information
      management, whole industries are developing and implementing
      standards for exchanging technical information. This report
      describes how one such standard, the Standard Generalized
      Markup Language (SGML), works as part of an overall
      information management strategy.</para>
      <graphic graphname="infoflow" />
    </intro>
  </chapter>
  <chapter number="2">
    <title>Getting to know SGML</title>
    <author>
      <last>Stevens</last>
      <first>W.</first>
    </author>
    <intro>
      <para>While SGML is a fairly recent technology, the use of
      <emph>markup</emph>in computer-generated documents has
      existed for a while.</para>
    </intro>
    <section shorttitle="What is markup?">
      <title>What is markup, or everything you always wanted to
      know about document preparation but were afraid to
      ask?</title>
      <intro>
        <para>Markup is everything in a document that is not
        content. The traditional meaning of markup is the manual
        <emph>marking</emph>up of typewritten text to give
        instructions for a typesetter or compositor about how to
        fit the text on a page and what typefaces to use. This kind
        of markup is known as
        <emph>procedural markup</emph>.</para>
      </intro>
      <topic id="top1" number="1">
        <title>Procedural markup</title>
```

```
      <keyword>Markup</keyword>
      <para security="u">Most electronic publishing systems today
      use some form of procedural markup. Procedural markup codes
      are good for one presentation of the information.</para>
    </topic>
    <topic id="top2" number="2">
      <title>Generic markup</title>
      <keyword>Markup</keyword>
      <para>Generic markup (also known as descriptive markup)
      describes the
      <emph>purpose</emph>of the text in a document. A basic
      concept of generic markup is that the content of a document
      must be separate from the style. Generic markup allows for
      multiple presentations of the information.</para>
    </topic>
    <topic id="top3" number="3">
      <title>Drawbacks of procedural markup</title>
      <keyword>Markup</keyword>
      <para security="u">Industries involved in technical
      documentation increasingly prefer generic over
      <emph>procedural</emph>markup schemes. When a company
      changes software or hardware systems, enormous data
      translation tasks arise, often resulting in errors.</para>
    </topic>
  </section>
  <section shorttitle="What is SGML?">
    <title>What
    <emph>is</emph>SGML in the grand scheme of the universe,
    anyway?</title>
    <intro>
      <para>SGML defines a strict markup scheme with a syntax for
      defining document data elements and an overall framework
      for marking up documents.</para>
      <para>SGML can describe and create documents that are not
      dependent on any hardware, software, formatter, or
      operating system. Since SGML documents conform to an
      international standard, they are portable.</para>
    </intro>
  </section>
  <section shorttitle="How does SGML work?">
    <title>How is SGML and would you recommend it to your
    grandmother?</title>
    <intro>
      <para>You can break a typical document into three layers:
      structure, content, and style. SGML works by separating
      these three aspects and deals mainly with the relationship
      between structure and content.</para>
    </intro>
    <topic id="top4" number="4">
      <title>Structure</title>
      <keyword>DTD</keyword>
      <para>At the heart of an SGML application is a file called
      the DTD, or Document Type Definition. The DTD sets up the
      structure of a document, much like a database schema
      describes the types of information it handles.</para>
      <para security="u">A database schema also defines the
      relationships between the various types of data. Similarly,
      a DTD specifies
```

```
        <emph>rules</emph>to help ensure documents have a
        consistent, logical structure.</para>
      </topic>
      <topic id="top5" number="5">
        <title>Content</title>
        <keyword>Markup</keyword>
        <para>Content is the information itself. The method for
        identifying the information and its meaning within this
        framework is called
        <emph>tagging</emph>. Tagging must conform to the rules
        established in the DTD (see
        <xref idref="top4" />).</para>
        <graphic graphname="tagexamp" />
      </topic>
      <topic id="top6" number="6">
        <title>Style</title>
        <para>SGML does not standardize style or other processing
        methods for information stored in SGML.</para>
      </topic>
    </section>
  </chapter>
  <chapter number="3">
    <title>Resources</title>
    <author>
      <last>Abiteboul</last>
      <first>Serge</first>
    </author>
    <author>
      <last>Buneman</last>
      <first>Peter</first>
    </author>
    <author>
      <last>Suciu</last>
      <first>Dan</first>
    </author>
    <section>
      <title>Conferences, tutorials, and training</title>
      <intro>
        <para>The Graphic Communications Association has been
        instrumental in the development of SGML. GCA provides
        conferences, tutorials, newsletters, and publication sales
        for both members and non-members.</para>
        <para security="c">Exiled members of the former Soviet
        Union's secret police, the KGB, have infiltrated the upper
        ranks of the GCA and are planning the Final Revolution as
        soon as DSSSL is completed.</para>
      </intro>
    </section>
  </chapter>
</report>
```

Appendix G. Data-oriented XML

G.1. Prices.dtd

```
<!ELEMENT book (title, store+, tip?)>
<!ATTLIST book
 year CDATA #IMPLIED
 id ID #IMPLIED
 available CDATA #IMPLIED
>
<!ELEMENT price (#PCDATA)>
<!ELEMENT prices (book+)>
<!ELEMENT source (#PCDATA)>
<!ELEMENT store (source, price)>
<!ELEMENT tip (xref+)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT xref EMPTY>
<!ATTLIST xref
 idref IDREF #IMPLIED
>
```

G.2. Prices.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE prices SYSTEM "prices.dtd">
<prices>
  <book year="1992" id="b1">
    <title>Advanced Programming in the Unix environment</title>
    <store>
      <source>bstore2.example.com</source>
      <price>65.95</price>
    </store>
    <store>
      <source>bstore1.example.com</source>
      <price>65.95</price>
    </store>
    <store>
      <source>www.bol.com</source>
      <price>45.95</price>
    </store>
    <tip>
      <xref idref="b2" />
    </tip>
```

```
  </book>
  <book year="1994" id="b2">
    <title>TCP/IP Illustrated</title>
    <store>
      <source>bstore2.example.com</source>
      <price>65.95</price>
    </store>
    <store>
      <source>bstore1.example.com</source>
      <price>65.95</price>
    </store>
    <tip>
      <xref idref="b1" />
    </tip>
  </book>
  <book year="2000" id="b3">
    <title>Data on the Web</title>
    <store>
      <source>bstore2.example.com</source>
      <price>34.95</price>
    </store>
    <store>
      <source>bstore1.example.com</source>
      <price>39.95</price>
    </store>
  </book>
  <book year="1999" id="b4">
    <title>The Economics of Technology and Content for Digital
    TV</title>
    <store>
      <source>www.amazon.com</source>
      <price>34.95</price>
    </store>
    <store>
      <source>www.bol.com</source>
      <price>69.95</price>
    </store>
  </book>
  <book year="1982" available="no" id="b5">
    <title>Getting started with SGML</title>
    <store>
      <source>www.mystore.com</source>
      <price>14.95</price>
    </store>
    <store>
      <source>www.bol.com</source>
      <price>45.95</price>
    </store>
  </book>
  <book year="1980" available="no" id="b6">
    <title>Drawbacks of procedural markup</title>
    <store>
      <source>bstore2.example.com</source>
      <price>25.95</price>
    </store>
    <tip>
      <xref idref="b5" />
    </tip>
```

```
  </book>
  <book year="1950" available="no" id="b7">
    <title>Lords of the Ring</title>
    <store>
      <source>bstore1.example.com</source>
      <price>55.95</price>
    </store>
    <store>
      <source>bstore2.example.com</source>
      <price>65.95</price>
    </store>
    <store>
      <source>bstore3.example.com</source>
      <price>25.95</price>
    </store>
    <store>
      <source>www.bol.com</source>
      <price>45.95</price>
    </store>
    <store>
      <source>www.amazon.com</source>
      <price>55.95</price>
    </store>
  </book>
  <book year="2003" id="b8">
    <title>XQuery from the experts</title>
    <store>
      <source>bstore2.example.com</source>
      <price>55.95</price>
    </store>
    <store>
      <source>www.amazon.com</source>
      <price>50.95</price>
    </store>
    <tip>
      <xref idref="b9" />
    </tip>
  </book>
  <book year="1999" id="b9">
    <title>XSLT: programmers reference</title>
    <store>
      <source>bstore2.example.com</source>
      <price>35.95</price>
    </store>
    <store>
      <source>www.amazon.com</source>
      <price>40.95</price>
    </store>
    <tip>
      <xref idref="b8" />
      <xref idref="b5" />
    </tip>
  </book>
</prices>
```

Appendix H. Results for Assignments

H.1. Introduction

In this experiment, we tested whether the use of a text-oriented XML document or a data-oriented XML documents leaded to differences between the user performance with XSLT and XQuery. The documents and DTDs can be found in Appendix F, *Document-oriented XML* and Appendix G, *Data-oriented XML*. We noticed that the factor document schema did not appear in any model as a significant predictor. Therefore, we excluded this factor from our analysis and combined the assignments with the same underlying query task to one assignment. In the following sections we show the results when XML document schema is included as a predictor. Note that the R Square values are generally lower compared to the analysis where we summarized the data regardless document schema. Also note that the results are very similar.

H.2. XSLT and XQuery Performance

The first research question is whether XQuery performance is better than XSLT performance. Again, we used a dependent samples t-test to check for a possible difference between mean number of subjects for each assignment between XSLT and XQuery. We considered only assignments when at least one subject attempted to complete the assignment for both XSLT and XQuery.

H.2.1. Correctness

In the following table the estimated mean values for correctness with XSLT and XQuery on the 128 assignments are presented. Note that 94 assignments were included for XSLT and 96 for XQuery, because only these assignments were attempted by at least one subject using XQuery or XSLT. The estimated mean values are corrected for the influences of the following factors: hours of XSLT experience, hours of XQuery experience, number of courses, average score on courses, number of programming courses, average score on programming courses, general experience of the subjects.

Table H.1. Correctness

Language	Mean	(Sem.)	N
XSLT	.56	(.051)	94
XQuery	.79	(.050)	96

An analysis of covariance showed that this difference is significant: $F(1,181) = 6.886$, $p < .009$. Therefore, XQuery has more correct queries than XSLT.

H.2.2. Time

In the following table the estimated mean values for time with XSLT and XQuery on the 128 assignments are presented. The performance measurement time is only meaningful for correctly solved queries: an incorrect query can be wrong after 30 minutes or after 10 seconds. Therefore, we only considered assignments that were the correctness is higher than zero. This was true for 76 assignments with XSLT and 87 assignments with XQuery. The estimated mean values are corrected for the influences of the following factors: hours of XSLT experience, hours of XQuery experience, number of courses, average score on courses, number of programming courses, average score on programming courses, general experience of the subjects.

Table H.2. Time

Language	Mean	(Sem.)	N
XSLT	528	(73,104)	76
XQuery	372	(65,552)	87

This difference is not significant: $F(1, 154) = 1.56$, $p < .21$. Therefore, there are no differences between XQuery and XSLT regarding query solving time.

H.3. Performance predictors

Stepwise multiple regression analysis was performed to explain the variance in average correctness and time on the query tasks. The following task and language related predictors were included in the analysis: XML document schema, task complexity, verboseness, number of expressions, number of Xpath expressions, Xpath length. In addition, the following user related predictors were included: hours of XSLT experience, hours of XQuery experience, number of participated courses, average score on all participated courses, number of participated programming courses, average score on all participated programming language courses, and general experience of the subjects. We show the results of the regression analyses in the following sections.

H.3.1. Correctness

For XSLT, we found that approximately 40% of the variance of correctness (R^2 = .398, F(5,88) = 11.632, p < .0005) was explained with predictors that are presented in the table below. For XQuery, we found that approximately 23% of the variance of correctness (R^2 = .234, F(2,92) = 14.228, p < .0005) was explained with the predictors that presented below. The predictors are ordered by their beta values, that is, for each language the predictors are ranked according to their importance.

Table H.3. Predictors for correctness

XSLT (R^2 = .398)	**XQuery (R^2 = .248)**
Number of programming courses (Beta = 1.226, sig = .000)	Task complexity (Beta = -.392, sig = .000)
Number of courses (Beta = -.905, sig = .001)	Number of programming courses (Beta = .288, sig = .002)
Task complexity (Beta = -.574, sig = .000)	
Average score on all course (Beta = .326, sig = .001)	
Xpath length (Beta = .265, sig = .035)	

The following predictors were not significant in the analyses of both XSLT and XQuery correctness: XML document schema, verboseness, number of expressions, number of Xpath expressions, hours of XSLT experience, hours of XQuery experience, average score on all participated programming language courses, and general experience of the subjects. In addition, for XQuery correctness the following predictors were not significant: number of participated courses, average score on all participated courses, Xpath length.

H.3.2. Time

For XSLT, we found that approximately 32% of the variance of time (R^2 = .317, F(2,73) = 16.933, p < .0005) was explained with predictors that are presented in the table below. For XQuery, we found that approximately 30% of the variance of correctness (R^2 = .304, F(2,84) = 18.327, p < .0005) was explained with the predictors that presented below. The predictors are ordered by their beta values, that is, for each language the predictors are ranked according to their importance.

Table H.4. Predictors for time

XSLT (R^2 = .317)	**XQuery (R^2 = .306)**
Xpath length (Beta = .514 sig = .000)	Verboseness (Beta = .705, sig = .000)
Number of programming courses (Beta = -.294, sig = .003)	Number of keywords (Beta = -.264, sig = .044)

The following predictors were not significant in the analyses of both XSLT and XQuery time: XML document schema, number of expressions, number of Xpath expressions, hours of XSLT experience, hours of XQuery experience, average score on all participated programming language courses, and general experience of the subjects. In addition, for XQuery correctness the following predictors were not significant: number of participated courses, average score on all participated courses, Xpath length.

Index

Samenvatting

XML (eXtensible Markup Language) is een coderingstaal die het mogelijk maakt om informatie voor hergebruik geschikt te maken. Om hergebruik eenvoudiger te maken zijn specifieke zoektalen voor XML ontworpen. Er zijn grote verschillen tussen deze talen met betrekking tot ontstaansgeschiedenis, beoogde doelen en syntax. Desondanks kunnen en worden zoektalen voor XML in de praktijk voor vergelijkbare doeleinden gebruikt. De motivatie voor het ontwerp van een zoektaal is vaak onduidelijk. Ook is onduidelijk wat het effect van een bepaalde keuze is op de effectiviteit waarmee een gebruiker met de taal kan werken.

In deze studie wordt de bruikbaarheid van de meest belangrijke zoektalen voor XML onderzocht. Onder bruikbaarheid wordt verstaan: de prestaties en de tevredenheid van gebruikers gemeten over een bepaalde set van taken. Er is nog weinig bekend over de bruikbaarheid van formele talen. In hoofdstuk 2 wordt een overzicht gegeven van eerder onderzoek naar de bruikbaarheid van database zoektalen. Hierbij komen o.a. eerder gebruikte onderzoeksmethodes en gebruikersmodellen aan bod.

De belangrijkste aanname in deze studie is dat verschil in syntax van een taal leidt tot verschil in bruikbaarheid. In hoofdstuk 2 wordt de achtergrond van deze aanname toegelicht. In de XML wereld zijn felle discussies gevoerd over de vermeende invloed van syntax op bruikbaarheid. Veelgehoorde argumenten in deze discussies zijn bijvoorbeeld dat XSLT lastig is in het gebruik vanwege o.a. de verbositeit van deze taal en dat XQuery prettiger is voor gebruikers met een database achtergrond vanwege de compacte syntax die deels gebaseerd is op populaire database zoektalen.

Prestatieverschillen tussen gebruikers van XML zoektalen moeten gerelateerd zijn aan hoe gebruikers vragen formuleren met een zoektaal. Als gebruikers met de ene taal meer tijd nodig hebben, of moeizamer tot een correcte zoekopdracht komen dan met een andere taal, dan verloopt het formuleren van een zoekopdracht bij de ene taal blijkbaar anders dan met de andere taal. In hoofdstuk 3 wordt een model voorgesteld hoe gebruikers vragen formuleren met een zoektaal.

Dit model wordt gebruikt om het gedrag te interpreteren van proefpersonen in een hardopdenk experiment. Op deze wijze wordt een eerste indruk verkregen van de bruikbaarheid van drie verschillende XML zoektalen (XSLT, XQuery en SQL/XML) en de oorzaken hiervan. Uit het experiment blijkt ook dat het model een adequate representatie van het formuleringsproces van een zoekopdracht is. Dit experiment wordt beschreven in hoofdstuk 4.

De verschillen in bruikbaarheid en de mogelijke factoren die hierbij van invloed zijn, worden verder onderzocht in een tweede experiment. Met een grote groep proefpersonen en een set van vijf zoekopdrachten wordt aangetoond dat gebruikers

beter presteren met XQuery en ook meer tevreden over deze taal zijn in vergelijking met XSLT. Dit experiment wordt beschreven in hoofdstuk 5.

Teneinde de precieze oorzaken van de bruikbaarheidsverschillen tussen XSLT en XQuery te verklaren wordt een derde experiment gehouden. Als mogelijke verklaringen voor prestatieverschillen wordt vooraf gedacht aan de complexiteit van de zoekvragen, de structuur van de XML documenten, de verbositeit van taalexpressies, de verschillende manieren waarop Xpath ingebed is in XSLT en XQuery, gebruikerservaring, en het aantal expressies dat nodig is voor een specifieke zoekvraag. Deze factoren worden gekwantificeerd bij een grote verzameling van vragen en de invloed van deze factoren op de prestaties wordt vastgesteld. Dit experiment wordt beschreven in hoofdstuk 6.

In deze studie wordt aangetoond dat de bruikbaarheid van XQuery groter is dan van XSLT. XQuery is gemakkelijker in het gebruik doordat de expressies korter zijn vergeleken met XSLT en omdat Xpath in XQuery op een eenvoudiger wijze wordt ingebed dan in XSLT. Ook is gebleken dat de ontwikkelde maat voor de complexiteit van de zoekopdracht een goede voorspeller is voor de gebruikersprestaties op een bepaalde zoekvraag en dat ook gebruikerservaring een grote rol speelt bij het voorspellen van gebruikersprestaties. De invloed van verschillende XML-structuren is niet aangetoond in deze studie. De vragen en de proefpersonen in deze studie zijn representatief voor het begin van de leercurve van deze talen. De conclusies van deze studie staan vermeld in hoofdstuk 7.

Curriculum Vitae

Joris Graaumans

November 20, 1974	Born in Eindhoven, The Netherlands.
1987 - 1994	High School, Van der Putt Lyceum & Eckart college, Eindhoven.
1994 - 1995	Study Social Worker, Fontys Hogeschool, Eindhoven.
1995 - 1999	Study Dutch Language and Literature, Utrecht University.
1999 - 2000	Technical writer, Windowsoft Europe BV, Eindhoven.
2000 - 2005	Ph.D. student at the Institute for Information and Computer Science, Faculty of Mathematics and Computer Science, Utrecht University.
2005	Information analyst, Tedopres BV, Tilburg.

SIKS Dissertations Series

1998

1998-1 Johan van den Akker (CWI): DEGAS - An Active, Temporal Database of Autonomous Objects

1998-2 Floris Wiesman (UM): Information Retrieval by Graphically Browsing Meta-Information

1998-3 Ans Steuten (TUD): A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspective

1998-4 Dennis Breuker (UM): Memory versus Search in Games

1998-5 E.W.Oskamp (RUL): Computerondersteuning bij Straftoemeting

1999

1999-1 Mark Sloof (VU): Physiology of Quality Change Modelling; Automated modelling of Quality Change of Agricultural Products

1999-2 Rob Potharst (EUR): Classification using decision trees and neural nets

1999-3 Don Beal (UM): The Nature of Minimax Search

1999-4 Jacques Penders (UM): The practical Art of Moving Physical Objects

1999-5 Aldo de Moor (KUB): Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems

1999-6 Niek J.E. Wijngaards (VU): Re-design of compositional systems

1999-7 David Spelt (UT): Verification support for object database design

1999-8 Jacques H.J. Lenting (UM): Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation

2000

2000-1 Frank Niessink (VU): Perspectives on Improving Software Maintenance

2000-2 Koen Holtman (TUE): Prototyping of CMS Storage Management

2000-3 Carolien M.T. Metselaar (UVA): Sociaal-organisatorische gevolgen van kennistechnologie; een procesbenadering en actorperspectief.

2000-4 Geert de Haan (VU): ETAG, A Formal Model of Competence Knowledge for User Interface Design

2000-5 Ruud van der Pol (UM): Knowledge-based Query Formulation in Information Retrieval

2000-6 Rogier van Eijk (UU): Programming Languages for Agent Communication

2000-7 Niels Peek (UU): Decision-theoretic Planning of Clinical Patient Management

2000-8 Veerle Coup, (EUR): Sensitivity Analyis of Decision-Theoretic Networks

2000-9 Florian Waas (CWI): Principles of Probabilistic Query Optimization

2000-10 Niels Nes (CWI): Image Database Management System Design Considerations, Algorithms and Architecture

2000-11 Jonas Karlsson (CWI): Scalable Distributed Data Structures for Database Management

2001

2001-1 Silja Renooij (UU): Qualitative Approaches to Quantifying Probabilistic Networks

2001-2 Koen Hindriks (UU): Agent Programming Languages: Programming with Mental Models

2001-3 Maarten van Someren (UvA): Learning as problem solving

2001-4 Evgueni Smirnov (UM): Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets

2001-5 Jacco van Ossenbruggen (VU): Processing Structured Hypermedia: A Matter of Style

2001-6 Martijn van Welie (VU): Task-based User Interface Design

2001-7 Bastiaan Schonhage (VU): Diva: Architectural Perspectives on Information Visualization

2001-8 Pascal van Eck (VU): A Compositional Semantic Structure for Multi-Agent Systems Dynamics

2001-9 Pieter Jan 't Hoen (RUL): Towards Distributed Development of Large Object-Oriented Models, Views of Packages as Classes

2001-10 Maarten Sierhuis (UvA): Modeling and Simulating Work Practice BRAHMS: a multiagent modeling and simulation language for work practice analysis and design

2001-11 Tom M. van Engers (VUA): Knowledge Management: The Role of Mental Models in Business Systems Design

2002

2002-01 Nico Lassing (VU): Architecture-Level Modifiability Analysis

2002-02 Roelof van Zwol (UT): Modelling and searching web-based document collections

2002-03 Henk Ernst Blok (UT): Database Optimization Aspects for Information Retrieval

2002-04 Juan Roberto Castelo Valdueza (UU): The Discrete Acyclic Digraph Markov Model in Data Mining

2002-05 Radu Serban (VU): The Private Cyberspace Modeling Electronic Environments inhabited by Privacy-concerned Agents

2002-06 Laurens Mommers (UL): Applied legal epistemology; Building a knowledge-based ontology of the legal domain

2002-07 Peter Boncz (CWI): Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications

2002-08 Jaap Gordijn (VU): Value Based Requirements Engineering: Exploring Innovative E-Commerce Ideas

2002-09 Willem-Jan van den Heuvel(KUB): Integrating Modern Business Applications with Objectified Legacy Systems

2002-10 Brian Sheppard (UM): Towards Perfect Play of Scrabble

2002-11 Wouter C.A. Wijngaards (VU): Agent Based Modelling of Dynamics: Biological and Organisational Applications

2002-12 Albrecht Schmidt (Uva): Processing XML in Database Systems

2002-13 Hongjing Wu (TUE): A Reference Architecture for Adaptive Hypermedia Applications

2002-14 Wieke de Vries (UU): Agent Interaction: Abstract Approaches to Modelling, Programming and Verifying Multi-Agent Systems

2002-15 Rik Eshuis (UT): Semantics and Verification of UML Activity Diagrams for Workflow Modelling

2002-16 Pieter van Langen (VU): The Anatomy of Design: Foundations, Models and Applications

2002-17 Stefan Manegold (UVA): Understanding, Modeling, and Improving Main-Memory Database Performance

2003

2003-01 Heiner Stuckenschmidt (VU): Ontology-Based Information Sharing in Weakly Structured Environments

2003-02 Jan Broersen (VU): Modal Action Logics for Reasoning About Reactive Systems

2003-03 Martijn Schuemie (TUD): Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy

2003-04 Milan Petkovic (UT): Content-Based Video Retrieval Supported by Database Technology

2003-05 Jos Lehmann (UVA): Causation in Artificial Intelligence and Law - A modelling approach

2003-06 Boris van Schooten (UT): Development and specification of virtual environments

2003-07 Machiel Jansen (UvA): Formal Explorations of Knowledge Intensive Tasks

2003-08 Yongping Ran (UM): Repair Based Scheduling

2003-09 Rens Kortmann (UM): The resolution of visually guided behaviour

2003-10 Andreas Lincke (UvT): Electronic Business Negotiation: Some experimental studies on the interaction between medium, innovation context and culture

2003-11 Simon Keizer (UT): Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks

2003-12 Roeland Ordelman (UT): Dutch speech recognition in multimedia information retrieval

2003-13 Jeroen Donkers (UM): Nosce Hostem - Searching with Opponent Models

2003-14 Stijn Hoppenbrouwers (KUN): Freezing Language: Conceptualisation Processes across ICT-Supported Organisations

2003-15 Mathijs de Weerdt (TUD): Plan Merging in Multi-Agent Systems

2003-16 Menzo Windhouwer (CWI): Feature Grammar Systems - Incremental Maintenance of Indexes to Digital Media Warehouses

2003-17 David Jansen (UT): Extensions of Statecharts with Probability, Time, and Stochastic Timing

2003-18 Levente Kocsis (UM): Learning Search Decisions

2004

2004-01 Virginia Dignum (UU): A Model for Organizational Interaction: Based on Agents, Founded in Logic

2004-02 Lai Xu (UvT): Monitoring Multi-party Contracts for E-business

2004-03 Perry Groot (VU): A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving

2004-04 Chris van Aart (UVA): Organizational Principles for Multi-Agent Architectures

2004-05 Viara Popova (EUR): Knowledge discovery and monotonicity

2004-06 Bart-Jan Hommes (TUD): The Evaluation of Business Process Modeling Techniques

2004-07 Elise Boltjes (UM): Voorbeeldig onderwijs; voorbeeldgestuurd onderwijs, een opstap naar abstract denken, vooral voor meisjes

2004-08 Joop Verbeek(UM): Politie en de Nieuwe Internationale Informatiemarkt, Grensregionale politiële gegevensuitwisseling en digitale expertise

2004-09 Martin Caminada (VU): For the Sake of the Argument; explorations into argument-based reasoning

2004-10 Suzanne Kabel (UVA): Knowledge-rich indexing of learning-objects

2004-11 Michel Klein (VU): Change Management for Distributed Ontologies

2004-12 The Duy Bui (UT): Creating emotions and facial expressions for embodied agents

2004-13 Wojciech Jamroga (UT): Using Multiple Models of Reality: On Agents who Know how to Play

2004-14 Paul Harrenstein (UU): Logic in Conflict. Logical Explorations in Strategic Equilibrium

2004-15 Arno Knobbe (UU): Multi-Relational Data Mining

2004-16 Federico Divina (VU): Hybrid Genetic Relational Search for Inductive Learning

2004-17 Mark Winands (UM): Informed Search in Complex Games

2004-18 Vania Bessa Machado (UvA): Supporting the Construction of Qualitative Knowledge Models

2004-19 Thijs Westerveld (UT): Using generative probabilistic models for multimedia retrieval

2004-20 Madelon Evers (Nyenrode): Learning from Design: facilitating multidisciplinary design teams

2005

2005-01 Floor Verdenius (UVA): Methodological Aspects of Designing Induction-Based Applications

2005-02 Erik van der Werf (UM):): AI techniques for the game of Go

2005-03 Franc Grootjen (RUN): A Pragmatic Approach to the Conceptualisation of Language

2005-04 Nirvana Meratnia (UT): Towards Database Support for Moving Object data

2005-05 Gabriel Infante-Lopez (UVA): Two-Level Probabilistic Grammars for Natural Language Parsing

2005-06 Pieter Spronck (UM): Adaptive Game AI

2005-07 Flavius Frasincar (TUE): Hypermedia Presentation Generation for Semantic Web Information Systems

2005-08 Richard Vdovjak (TUE): A Model-driven Approach for Building Distributed Ontology-based Web Applications

2005-09 Jeen Broekstra (VU): Storage, Querying and Inferencing for Semantic Web Languages

2005-10 Anders Bouwer (UVA): Explaining Behaviour: Using Qualitative Simulation in Interactive Learning Environments

2005-11 Elth Ogston (VU): Agent Based Matchmaking and Clustering - A Decentralized Approach to Search

2005-12 Csaba Boer (EUR): Distributed Simulation in Industry

2005-13 Fred Hamburg (UL): Een Computermodel voor het Ondersteunen van Euthanasiebeslissingen

2005-14 Borys Omelayenko (VU): Web-Service configuration on the Semantic Web; Exploring how semantics meets pragmatics

2005-15 Tibor Bosse (VU): Analysis of the Dynamics of Cognitive Processes

2005-16 Joris Graaumans (UU): Usability of XML Query Languages