

# Xcerpt and visXcerpt: From Pattern-Based to Visual Querying of XML and Semistructured Data

Sacha Berger, François Bry, Sebastian Schaffert, Christoph Wieser

Institut für Informatik, Ludwig-Maximilians-Universität München

## 1 Overview

With the advent of XML as a format for data exchange and semistructured databases, query languages for XML and semistructured data have become increasingly popular.

Many such query languages, like XPath and XQuery, are *navigational* in the sense that their variable binding paradigm requires the programmer to specify path navigations through the document (or data item). In contrast, some other languages – such as UnQL [1] and Xcerpt [2] – are pattern-based: their variable binding paradigm is that of mathematical logics, i.e. the programmer specifies patterns (or terms) including variables. Arguably, a pattern-based variable binding paradigm makes complex queries much easier to specify and to read, thus improving the programming efficiency. Sustaining this first claim with practical examples is one of the objectives of the present demonstration.

Xcerpt [2] is an experimental pattern-based query and transformation language for XML and semistructured data. Xcerpt uses patterns both for binding variables in query expressions and for reassembling the variables (bound to data items in query expressions) in so-called *construct terms*. Arguably, a pattern-based document construction combined with a pattern-based variable binding results in a rather intuitive, user friendly, and programming efficient language. Sustaining this second claim is another objective of the present demonstration.

Xcerpt is experimental in the sense that its purpose is to investigate and test another, non-navigational approach to retrieve data from the Web than that of the widespread query languages XPath and XQuery. Nonetheless, Xcerpt has been prototypically imple-

mented and is currently experimented with on test bed examples for proof-of-concept purposes.

Another important characteristic of Xcerpt is its rule-based nature: Xcerpt provides with rules very similar to SQL view definitions. Arguably, rules or views are convenient for a logical structuring of complex queries. Thus, in specifying a complex query, it might ease the programming and improve the program readability to specify (abstract) rules as intermediate steps – very much like procedures are used in conventional programming for factorising out common computations and/or modularising a program. Sustaining this third claim is a further objective of the present demonstration.

Because of its rule-based nature, Xcerpt has been selected as a contribution to the RuleML initiative [3], an international tentative to promote the interoperability of rule and/or reasoning based languages for the (Semantic) Web.

Based on Xcerpt, a *visual query language* called *visXcerpt* has been conceived and prototypically implemented. Most interestingly, the pattern-based approach to query and transformation languages for XML and semistructured data appears to be especially well suited for a visual (or graphical) language. The reason is that patterns are form-like two dimensional structures that conceptually are very close to two dimensional visual representations. Arguably, every visual or graphical language for XML and/or semistructured data (such as XML-GL [4], GraphLog [5], VXT [6], BBQ [7] and Xing [8]) as well as the veteran language QBE and improvements thereof (such as MS Access and similar products) might be seen as having an (in general implicit) pattern-based language as an (in general unconscious) foundation. Sustaining this fourth claim is a last objective of the present demonstration.

Interestingly, and maybe supporting the last above-mentioned claim, a visual language for a pattern-based textual query and transformation language can be developed simply by specifying a visual *rendering* (in contrast to a complex transformation) of the textual programs very much like a CSS stylesheet specifies a layout for an HTML document.

---

*Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, requires a fee and/or special permission from the Endowment.*

**Proceedings of the 29th VLDB Conference,  
Berlin, Germany, 2003**

## 2 Positional vs. Navigational

Essential to querying semistructured data is the selection of data items in a document (i.e. rooted graph). Most widespread query languages for XML – e.g. XQuery – rely on path selections expressed using XPath (or similar approaches). XPath-like languages provide with constructs like regular expressions and wild cards for specifying paths through a rooted graph. For instance, the XPath expression `/a[b]//c` means “find the document nodes labelled *c* that can be reached from the document root via a child node labelled *a* having itself a child node labelled *b* and having the *c*-labelled nodes as descendants”. Such node selections can be called *navigational*.

For simple queries and transformations, the navigational approach is very natural and results in simple programs. For more complex queries, especially for queries involving several variables, the navigational approach often leads to intricate programs.

Furthermore, the intertwining of construction and query parts in languages such as XQuery and most of its precursors [9] often yields programs that are difficult to read – and hence to use and to maintain.

Also, the possibility to specify forward and reverse axes in path languages like XPath might further increase the complexity of query programs, while the intuitive meaning is often not complicated at all.

A further important aspect of navigational node selections is that they do not easily support the selection of *several* related nodes at once. Such multiple node selections, however, are rather natural and are required by most non-trivial queries. This is e.g. the case when one looks for bibliography entries combining several aspects such as an author’s name, a keyword in the title, and a year of publication. Everyone familiar with bibliographies immediately “visualises” the shape or pattern of such a retrieval request and the respective positions of the variables it refers to. Pattern-based or *positional* query and transformation languages such as Xcerpt reflect and convey such an intuitive “visualisation”.

With the *positional* query and transformation language Xcerpt the nodes to be selected are specified by variables in patterns called *query terms*. Query patterns are related to other patterns called *construct terms* through their common variables. The Xcerpt construct relating a construct term to a query expression consisting of AND and/or OR connected query terms is a *rule*. Xcerpt has been introduced in [2].

For querying semistructured data, the positional approach has been suggested first with UnQL [1] and XML-QL [10]. In common programming, the positional approach finds its roots in Functional and Logic Programming. Also, both query languages QBE and SQL can be seen as positional languages.

## 3 Xcerpt Main Constructs

An Xcerpt program may consist of at least one *goal* and of some (maybe zero) *rules*. Goals and rules are built up from database, query and construct terms that are first introduced. Note that besides the “abstract” syntax presented here, Xcerpt also has an XML syntax which is not described here for space reasons.

### 3.1 Database, Query, and Construct Terms

Common to all terms is that they represent tree-like (or graph-like) structures. The children of a node may be either *ordered* (as in standard XML) or *unordered* (as is common in databases).

Square brackets denote *ordered term specification*, i.e. the matching subterms in the database are required to be in the same order as in the query term. Curly braces denote *unordered term specification*, i.e. the matching subterms in the database may be in arbitrary order.

Single (square or curly) braces are used to denote that a matching term must contain matching subterms for all subterms of a term and may not contain additional subterms (*total term specification*). Double braces are used to denote that the database term may contain additional subterms as long as matching partners for all subterms of the query term are found (*partial term specification*).

**Database terms** are used to represent XML documents and the data items of a semistructured database. They are similar to *ground* functional programming expressions and logical atoms. A *database* is a (multi-)set of database terms (e.g. the Web).

*Example:* a book database

```
bib { book {
  title { "Data on the Web" },
  authors [ author { "Se.g. Abiteboul" },
            author { "Peter Buneman" },
            author { "Dan Suciu" } ],
  price { "69.95" } }, ...
}
```

**Query terms** are similar to *non-ground* functional programming expressions and logical atoms. In contrast to functional expressions and logical atoms, query terms have the following properties:

- in a query term, partial specifications omitting subterms irrelevant to the query are possible,
- in a query term like in a database term, subterms might be ordered or unordered,
- in a query term, it is possible to specify subterms at arbitrary depth.

*Example:* This query term selects titles and authors:

```
bib {{ book {{
  var T ~ title, authors {{ var A }}
}} }}
```

Query terms are *unified* with database or construct terms using a non-standard unification called *simulation unification* [11]. It is based on *graph simulation* [12] which is similar to graph homomorphisms.

The outcome of unifying a query term with a database term are bindings for the variables in the query term. Applying these bindings to the query term results in a ground query term which is simulated (in the sense of [12]) in the database term.

The Xcerpt construct  $X \rightsquigarrow t$  (read “as”) serves to associate a query term to a variable, so as to specify a restriction of its bindings. The Xcerpt construct *desc* (read “descendant” – not illustrated above) is used to specify subterms at arbitrary depth.

**Construct terms** serve to reassemble variable (the bindings of which are specified in query terms) so as to construct new database terms. In a construct term,  $\{ \}$  and  $[ ]$  as well as variables might occur but  $\rightsquigarrow$  are precluded. The rationale of this restriction is to keep variable specifications within query terms, ensuring a clear separation of purposes between query and construct terms.

In a construct term, the Xcerpt construct *all*  $t$  serves to collect (in the construct term) all instances of  $t$  that can be generated by different variable bindings for the variables in  $t$  (returned by the associated query terms in which they occur).

*Example:* The following construct term collects all title/author pairs for the previous query:

```
results { all result { var T, var A } }
```

### 3.2 Construct-Query Rules

An Xcerpt construct-query rule (short: rule) relates a construct term to a query consisting of AND and/or OR connected query terms. An Xcerpt rule has the form  $C \leftarrow Q$  where  $C$  denote a construct term and  $Q$  denotes a query.

This rule can be seen as a “view” specifying how  $C$ -shaped documents can be obtained by evaluating the query  $Q$  against a Web resource (e.g. an XML document or a database).

An Xcerpt query might contain one or several references to *resources*. Xcerpt rules might be *chained* like active or deductive database rules to form complex query programs.

### 3.3 Further Constructs

The previous section only describes Xcerpt’s core constructs. In addition, Xcerpt offers the following advanced constructs:

- a reference mechanism to build graph structures instead of tree structures
- *some* for a non-deterministic selection of some variable instances in a construct term
- elementary arithmetics and string operations



Figure 1: A rule in Xcerpt’s visual syntax

- aggregation constructs

## 4 visXcerpt: A Visual Rendering of Xcerpt

Essential to the visual language visXcerpt is that Xcerpt is based on (query and construct) terms specifying for each occurrence of a variable its *position* in a query or construct expression instead of a *navigation* path (from the document root to the variable occurrence). As a consequence, textual Xcerpt’s visual counterpart *visXcerpt* can be conceived as a mere *rendering* instead of a fully novel language. This rendering might be seen as an advanced (because of the dynamic features) layout. VisXcerpt is realized using CSS stylesheets for the static layout and ECMA Script for the dynamic features.

Xcerpt **terms** (i.e. elements) are visualised as boxes. A term label (or tag) is attached as a tab on the top of its associated box. visXcerpt has features for handling attributes and text. Attributes are placed in a two-column table with names in the left column and values in the right column. The attribute table appears first in a box and is omitted if there are no attributes. Direct subterms (i.e. children) are visualised the same way as sub- or child boxes. Child boxes are arranged vertically in a parent box. For better distinction, they are coloured differently.

Different box borders are used as visual counterparts to the **Xcerpt parentheses**  $\{ \}$ ,  $[ ]$ ,  $\{ \}$ , and  $[ ]$ . Ordered or unordered children are indicated graphically by an icon at the top right corner of a box.

Optionally, partial and total matching is also graphically indicated by an icon.

Xcerpt constructs like *desc* (descendant), *all* and *variables* are represented as boxes with different, reserved colours in some cases (e.g. *all*) with a textual adornment. Variables are visualised as black boxes with the variable name written in white in the box. If a variable is restricted to a term (by the  $\rightsquigarrow$  construct), this term appears within the box of the variable.

**Construct-query rules** are visualised as shown in Figure 1.

**Dynamic features.** By clicking on a box tab (representing a term label or tag), the associated box is *folded* resp. *unfolded*. VisXcerpt programs can be *edited* using copy-and-paste-based primitives accessible from context menus. For constructing new contents, a document offering all constructs of visXcerpt serves as a copy template. By moving the mouse pointer across a variable name, the boxes representing other occurrences of the same variable are highlighted. References in terms will be implemented in visXcerpt using the same highlighting. Furthermore, references will be traversable as hyperlinks.

## 5 Outline of the Demonstration

The demonstration presents Xcerpt and visXcerpt prototypes running on Linux using the Apache web server and the Mozilla web browser.

The **Xcerpt prototype** is implemented in Haskell. It can process Xcerpt programs in both Xcerpt and XML syntax. It can access XML, Xcerpt, and HTML resources as local files or on the Web (via the hypertext transfer protocol HTTP).

The **visXcerpt prototype** is implemented mainly in CSS. Its dynamic aspects (like program editing) are implemented in ECMA Script. The visXcerpt prototype renders programs into a dynamic HTML representation which can be displayed using a standard compliant web browser (Mozilla is chosen for the demonstration). visXcerpt programs are evaluated as follows: the visual program is translated into an (XML) textual program which is sent for evaluation (via HTTP) to the (textual) Xcerpt prototype.

In the demonstration we first present the textual language Xcerpt, then the visual language visXcerpt.

**Presentation of Xcerpt.** The Xcerpt presentation begins with a single, very simple rule. Building on this rule, more complex transformations are created using the different positions for the all-construct and different variable restrictions. It is then demonstrated how to combine several queries with multiple occurrences of the same variable. Finally, two additional rules are added to illustrate how several rules can be used to nicely separate program logic from result presentation.

**Presentation of visXcerpt.** The presentation of the visual interface begins with illustrating how

generic XML documents and the Xcerpt attributions (order/partiality) are represented visually. Concepts like browsing and editing primitives are introduced. The visual rendering of Xcerpt rules and Xcerpt constructs is then illustrated on example programs.

## References

- [1] Buneman, P., Fernandez, M., Suciu, D.: UnQL: A Query Language and Algebra for Semistructured Data Based on Structural Recursion. VLDB Journal **9** (2000) 76–110
- [2] Bry, F., Schaffert, S.: A Gentle Introduction into Xcerpt, a Rule-based Query and Transformation Language for XML. In: Proc. Int. Workshop on Rule Markup Languages for Business Rules on the Semantic Web. (2002) (invited article).
- [3] RuleML Initiative <http://www.dfki.uni-kl.de/ruleml/>: Rule Markup Language. (2002)
- [4] Ceri, S., Damiani, E., Fraternali, P., Paraboschi, S., Tanca, L.: XML-GL: A Graphical Language for Querying and Restructuring XML Documents. In: Sistemi Evoluti per Basi di Dati. (1999)
- [5] Consens, M., Mendelzon, A.: Expressing Structural Hypertext Queries in GraphLog. In: Second ACM Hypertext Conf. (1989) 269–292
- [6] Pietriga, E., Quint, V., Vion-Dury, J.Y.: VXT: A Visual Approach to XML Transformations. In: ACM Symp. on Document Engineering. (2001)
- [7] Munroe, K.D., Papakonstantinou, Y.: BBQ: A Visual Interface for Integrated Browsing and Querying of XML. In: VDB. (2000)
- [8] Erwig, M.: A Visual Language for XML. In: IEEE Symp. on Visual Languages. (2000) 47–54
- [9] Maier, D.: Database Desiderata for an XML Query Language. In: Proc. of QL'98 - The Query Languages Workshop. (1998)
- [10] Deutsch, A., Fernandez, M., Florescu, D., Levy, A., Suciu, D.: A Query Language for XML. In: Proc. of Eighth Int. WWW Conf. (1999)
- [11] Bry, F., Schaffert, S.: Towards a Declarative Query and Transformation Language for XML and Semistructured Data: Simulation Unification. In: Proc. Int. Conf. on Logic Programming (ICLP). LNCS 2401, Springer-Verlag (2002)
- [12] Abiteboul, S., Buneman, P., Suciu, D.: Data on the Web. From Relations to Semistructured Data and XML. Morgan Kaufmann (2000)