I4-D6

Initial Draft of a Language Syntax

Project title:	Reasoning on the Web with Rules and Semantics
Project acronym:	REWERSE
Project number:	IST-2004-506779
Project instrument:	EU FP6 Network of Excellence (NoE)
Project thematic priority:	Priority 2: Information Society Technologies (IST)
Document type:	D (deliverable)
Nature of document:	R (report)
Dissemination level:	PU (public)
Document number:	IST506779/Munich/I4-D6/D/PU/a1
Responsible editors:	Tim Furche
Reviewers:	Liviu Badea and Gerd Wagner
Contributing participants:	Munich
Contributing workpackages:	I4
Contractual date of deliverable:	31 August 2005
Actual submission date:	20 January 2006

Abstract

This article defines an initial proposal for the syntax of the I4 query language, Xcerpt. Indeed, not only a single syntax, but rather three syntactical forms of Xcerpt are introduced: (1) the term syntax, a non-standard syntax that allows the succinct formulation of queries and is intended mostly for human authors; (2) the XML syntax provides a fine granular language markup in XML, ideal for processing through XML-based tools and for automated query generation or reasoning about query programs; (3) the compact XML syntax is a hybrid syntax of (1) and (2). The concepts are introduced UML. In addition to the formal syntax specification, principles of the syntax design are disucssed. Furthermore, for a number of advanced constructs the reasoning supporting the design choice, as well as alternative solutions are illustrated. An impression of how the introduced language constructs allow to write and understand complex queries is given by numerous examples interspersed among the construct specifications.

Keyword List

reasoning, query language, Semantic Web, concepts, grammar, syntax, terms, XML

Project co-funded by the European Commission and the Swiss Federal Office for Education and Science within the Sixth Framework Programme.

Initial Draft of a Language Syntax

François Bry[1], **Tim Furche**[1], **Sebastian Schaffert**[2]

[1] Institute for Informatics, University of Munich, Germany
Email: {Francois.Bry,Tim.Furche}@ifi.lmu.de

[2] Salzburg Research, Salzburg, Austria
Email: {Sebastian.Schaffert@salzburgresearch.at}

20 January 2006

Abstract
This article defines an initial proposal for the syntax of the I4 query language, Xcerpt. Indeed, not only a single syntax, but rather three syntactical forms of Xcerpt are introduced: (1) the term syntax, a non-standard syntax that allows the succinct formulation of queries and is intended mostly for human authors; (2) the XML syntax provides a fine granular language markup in XML, ideal for processing through XML-based tools and for automated query generation or reasoning about query programs; (3) the compact XML syntax is a hybrid syntax of (1) and (2). The concepts are introduced UML. In addition to the formal syntax specification, principles of the syntax design are disucssed. Furthermore, for a number of advanced constructs the reasoning supporting the design choice, as well as alternative solutions are illustrated. An impression of how the introduced language constructs allow to write and understand complex queries is given by numerous examples interspersed among the construct specifications.

Keyword List
reasoning, query language, Semantic Web, concepts, grammar, syntax, terms, XML

Contents

Chapter 1

Introduction

Xcerpt is *semi-structured query language*, but very much unique among the exemplars of that type of query languages (for an overview see [4]):

1. In its use of a **graph data model**, it stands more closely to early semi-structured query languages such as Lorel [1, 33] than to current mainstream XML query languages.
2. In its aim to address all specificities of **XML with great care**, it resembles more current mainstream XML query languages such as XSLT [13] or XQuery [6]. Xcerpt is tailored to XML in numerous ways, e.g., by proper support for attributes and namespaces [7]. This is achieved without sacrificing the conceptual simplicity and syntactical conciseness of the language. Some aspects of XML are treated differently than in mainstream XML query languages, e.g., the transparent resolution of non-hierarchical relations expressed using ID/IDREF, XLink [18], etc.
3. In using (slightly enriched) **patterns** (or templates or examples) of the sought-for data for querying, it resembles more the "query-by-example" paradigm [39] and XML query languages such as XML-QL [19]. In contrast, current mainstream XML query languages use navigational access to XML data.
4. In offering a **consistent extension of XML** to overcome certain restrictions of XML, that seem arbitrary in the context of Web querying and Xcerpt in particular, it is ready to incorporate access to data represented in richer data representation formats. Instances of such features are element content, where the *order is irrelevant* (and can not be queried) and labels that contain "reserved" XML characters.
5. In providing (syntactical) extensions for querying, among others, RDF, Xcerpt becomes a **versatile query language** (as defined in [10]).
6. In a strict separation of querying and construction and in its use of logical variables and deductive rules, it resembles more logic programming languages or Datalog. In contrast, SQL, e.g., mixes construction and querying (nested queries) and uses explicit references to views rather than rule chaining.

These unique characteristics of Xcerpt motivate many of the language concepts introduced in the remainder of this chapter, a more complete discussion of the guiding design principles for (versatile) Web query languages as exemplified in Xcerpt can be found in [10].

Xcerpt exhibits not just one concrete syntax, but rather three, each focused on providing a unique set of strengths.

1. The first, **non-XML syntax** is based around the idea of representing **terms as in logic-programming** and the following principles:

   (a) Terms are represented **similar to logic-programming**: prefix notation with bracketed argument lists for the children of the term. Special provisions are made to adapt this basic principle to handle the specificities of XML and other Web formats.

   (b) Different types of brackets encode term properties and distinguish language constructs from data.

   (c) The syntax strives to be *concise*, but still *easy to read*. The latter objective is supported, e.g., by the preference for explicit full-word keywords (e.g., `optional`) to represent language constructs instead of shorthand notations (such as `@` for attribute in XPath).

   (d) The non-XML term syntax emphasizes that Xcerpts data model and language features are not specific for *one* representation formalism such as XML and RDF, but rather allow different ones to be handled with the same concepts by mapping them to Xcerpt terms, still providing for all the specificities of the supported representation formalisms.

2. The **hybrid XML-style term syntax** is a rather recent development. It aims at

   (a) providing a syntax that is (almost) **immediately accessible** to persons accustomed with XML;

   (b) **very explicit**, i.e., uses in addition to the XML syntax uses only English words to represent language features;

   This makes the syntax both easier to read and harder to write, as it is slightly less compact than the term syntax but therefore often uses English words to represent language features instead of special symbols.

   The mixing of XML-style syntax for terms and keywords as in the non-XML syntax raises a number of potential clashes. Most notably, *character data still has to be quoted* in contrast to XML to avoid having to escape non-XML parts of the syntax.

3. The previous two syntaxes are mainly intended for human use. Like Relax NG, Xcerpt also exhibits a **pure XML syntax** that, though harder to author and read for humans, is easy to process with XML tools. The guiding principle is a form of *markup reification*, i.e., elements and attributes are explicitly represented by XML elements named `element` and `attribute` (similar to XSLT's `xsl:element` and `xsl:attribute` elements for construction of elements with computed names).

   As in the other concrete syntaxes, the Xcerpt namespace `http://xcerpt.org/ns/core/1.0` is reserved to indicate language constructs. Indeed, in the XML syntax *all* language constructs are expressed through elements in the Xcerpt namespace.

Chapter 2

Meta-Syntax Notations for Abstract and Concrete Syntax

This article presents the syntax of Xcerpt from three different perspectives: an abstract syntax focusing on the language concepts, a concrete syntax that represents terms in a compact style familiar from, e.g., logic programming, and a concrete XML syntax that represents a basis for Xcerpt tools and machine processing of Xcerpt programs.

To define each of these syntaxes appropriate meta-languages are chosen: For the abstract syntax of Xcerpt—in other words, its information model, what form of information is needed for which feature of the language—the OMG's Unified Modelling Language (UML) is used. The concrete term syntax is defined using EBNF grammars and/or railroad syntax diagrams. Finally, the concrete XML syntax is specified by means of Relax NG schemata.

The remainder of this chapter serves (1) as a (very brief) introduction in the notions of these meta-languages used in this article (2) to define, where necessary, the precise variant of the meta-languages referred to in the following chapters, and (3) to point to authoritative specifications of the meta-languages.

2.1 Abstract Syntax: UML Diagrams or What is the Information conveyed in an Xcerpt Program?

UML models are shown here in the notation from [35], the OMG UML 2.0 Superstructure specification. However, only a small subset of UML's diagrams and notions is needed for the purpose of this article: the abstract syntax is defined using *(static) class diagrams*. Moreover, neither attributes nor methods are present in the diagrams, thus allowing the diagrams to be considered as merely concepts and relations.

In particular, *generalizations* (solid line, with an closed, but unfilled arrow head at the end connected to the more general concept) are used to express different variants of a concept, e.g., the different kinds of data terms, each representing a more specialized variant of the general concept "data term". Generalizations can be decorated with constraints (attached to the line in braces). In the following, only the complete and the disjoint constraint are used. The first indicates that no instances of the more general concept exist, that are not also instance of

(at least) one of the specialized concepts, the latter that the instance sets of the specialized concepts are disjoint.

Aggregations (solid line with an unfilled rhombus at the end connected to the "whole" concept) are used for "part-whole" relations, e.g., to express that data term-level declarations contain data terms. Roles and multiplicities can be used to further annotate aggregations (and other relations).

One advanced concept from UML is used to highlight the differences and commonalities among the three kinds of Xcerpt terms: *"parameterized collaborations"*. UML uses parameterized collaborations to describe what is often referred to as (software) *patterns* (not to be confused with Xcerpt's patterns), i.e., collections of concepts and relations among concepts that occur in different contexts. They are "parameterized", as a number of the concepts in the parameterized collaboration are "exported" as parameters and must be related to concrete concepts when using the pattern. See Section 5.1 for a description of a model for Xcerpt terms based on parameterized collaborations.

For more information on UML (including tutorials) see the OMG UML homepage `http://www.uml.org/`.

2.2 Concrete Syntax: Regular Expression-Style EBNF or Defining the Textual Term Syntax for Xcerpt

A common choice to define the textual syntax of a programming or query language is a variant of the "Bachus Naur Form" introduced for the specification of ALGOL [3].

Several extensions, then referred to as Extended BNF or EBNF, to the basic BNF notation have been suggested, mostly adding some form of repetition and optionality to the original language (that only provided constructs for terminals, non-terminals, definition of non-terminals, and alternative).

Indeed, several standardization bodies have recently defined "standard' '[1] variants of BNF, most notably the ISO EBNF international standard [28], the IETF [17] internet standard, and the W3C-style EBNF notation defined in Section 6 of the XML specification [8]. Table 2.1 shows the constructs of the ISO EBNF standard in contrast to the constructs of the W3C-style EBNF given in Table 2.2.

For this article, the W3C-style notation is used, since it is reminiscent of regular expressions as also used in Xcerpt and is likely to be most familiar for readers accustomed to W3C standards.

Like [6], we slightly deviate from the syntax in [8] to address the differences in print vs.online publications:

1. Instead of marking non-terminal symbols with links (commonly displayed using underline and blue color), non-terminal symbols are enclosed in typographic angle brackets ($\langle\cdot\rangle$) and set in italics.

2. To further strengthen the difference between meta-language and language constructs, teletype is reserved for terminal symbols, all non-terminals and meta-language constructs are formatted as usual text. In particular, typographic single quotes are used for quoting non-terminals.

[1] Refined from the dozens of variants, cf. `http://www.cs.man.ac.uk/~pjj/bnf/ebnf.html`.

ISO EBNF Construct	Operator Type	Meaning
`unquoted word`		non-terminal symbol
`"..."`		terminal symbol
`'...'`		terminal symbol
`(...)`		grouping override precedence)
`[...]`		optional symbols
`{...}`		zero or more repetition of symbols
`{...}-`		one or more repetition of symbols
`=`	in	symbol definition
`;`	post	rule terminator
`|`	in	alternative
`,`	in	symbol concatenation
`-`	in	except
`*`	in	n occurrences of symbols
`(*...*)`		comment
`?...?`		special sequences (extensible)

Table 2.1: ISO Standard Extended BNF Notation

W3C EBNF Construct	Operator Type	Meaning
`unquoted word`		non-terminal symbol
`"..."`		terminal symbol
`'...'`		terminal symbol
`[...]`		character groups as in regular expressions
`(...)`		grouping (to override precedence)
`...?`	post	optional symbols
`...*`	post	zero or more repetition of symbols
`...+`	post	one or more repetition of symbols
`::=`	in	symbol definition
`|`	in	alternative
`-`	in	except
`/*...*/`		comment

Table 2.2: W3C Extended BNF Notation

3. Whitespace handling is left out of the grammar: By default, additional whitespace may occur anywhere between non-terminals. The few exceptions (names, IRIs, and strings) are noted in the natural language description of the grammar rules.

Finally, to reference non-terminals defined in other specifications a notation similar to the one proposed in [6] is adopted: ⟨*http://www.w3.org/TR/REC-xml-names/#Digit*⟩ references the non-terminal ⟨*Digit*⟩ from the XML specification identified through its canonical URI.

EBNF rules are often visualized using railroad syntax diagrams similar to [9]: terminals and non-terminals are drawn on a line to specify concatenation. Alternatives are represented by stacked lines fanning out at the decision point and repetition is indicated using loops (optionality comes for free as alternative where one of the stacked lines does not contain terminals or non-terminals).

2.3 Concrete Syntax: Relax NG or A Schema for Xcerpt Programs in XML

Xcerpt's XML syntax is specified in form of an Relax NG grammar. Relax NG [15] is a schema language for XML. It has been chosen for the specification of Xcerpt's syntax as it (a) has, in contrast to XML Schema [22], a compact, easy to read syntax, (b) supports, in constrast to DTDs namespaces, and (c) has support for parameterized grammar rules. The latter point makes it possible to drastically reduce the size of the language specification by reusing the definition of, e.g., a term over all three term kinds of Xcerpt, parameterizing where necessary. It also allows a close alignment with the abstract syntax.

Relax NG has, like Xcerpt, both a compact textual non-XML syntax [14] and a more verbose pure XML syntax. In the following, the compact non-XML syntax for Relax NG is used.

A Relax NG grammar consists in a single **`start`** production and a set of normal productions, each defining one non-terminal (called *named pattern*). Element content can be defined using connectives like in regular expressions or DTDs: sequence (`,`), choice (`|`), repetition (`*` and `+`), and optionality (`?`). Elements and attributes are treated symmetrically, specified using **`element`**, resp. **`attribute`** followed by the name of the element or attribute. Literal content is either **`text`** or content typed according to the XML Schema Datatypes [5].

Non-terminals (named patterns) may be defined by multiple productions, if each production is marked with the how to *combine* the alternatives: `|=` instead of the usual `=` indicates that the alternative productions form a choice. I.e., if `S |= P` and `S |= Q` then `S = P | Q`. This allows the introduction of additional choices, e.g., when including an existing grammar and is used extensively to define construct and query terms as extensions of the productions for data terms, cf. Chapter 5, and Chapter 6.

Relax NG is a particular convenient choice for defining Xcerpt, as it allows (to some extend) to express *parameterized concepts* (as discussed above): In Relax NG grammars (i.e., sets of productions or (non-terminal) definitions) may be *merged* or included into each other. When including one grammar in another one (using the **`include`** keyword and a reference to the file in which the grammar to be included is contained), definitions for non-terminals may be replaced or combined with new ones. Such replaced non-terminals can be seen like parameters of the grammar. Grammars can furthermore be nested to "hide" away all productions except the start symbol of the grammar.

Namespaces can be attached to all elements of a grammar or to individual elements. There only the first means are used, cf. Section 4.1.

More information about Relax NG and its compact syntax can be found at the OASIS Relax NG site `http://www.relaxng.org/`.

Part I

Definition of the Core Language

Chapter 3

Xcerpt: A Versatile Web Query Language

Xcerpt is *semi-structured query language*, but very much unique among the exemplars of that type of query languages (for an overview see [4]):

1. In its use of a **graph data model**, it stands more closely to early semi-structured query languages such as Lorel [1, 33] than to current mainstream XML query languages.

2. In its aim to address all specificities of **XML with great care**, it resembles more current mainstream XML query languages such as XSLT [13] or XQuery [6]. Xcerpt is tailored to XML in numerous ways, e.g., by proper support for attributes and namespaces [7]. This is achieved without sacrificing the conceptual simplicity and syntactical conciseness of the language. Some aspects of XML are treated differently than in mainstream XML query languages, e.g., the transparent resolution of non-hierarchical relations expressed using ID/IDREF, XLink [18], etc.

3. In using (slightly enriched) **patterns** (or templates or examples) of the sought-for data for querying, it resembles more the "query-by-example" paradigm [39] and XML query languages such as XML-QL [19]. In contrast, current mainstream XML query languages use navigational access to XML data.

4. In offering a **consistent extension of XML** to overcome certain restrictions of XML, that seem arbitrary in the context of Web querying and Xcerpt in particular, it is ready to incorporate access to data represented in richer data representation formats. Instances of such features are element content, where the *order is irrelevant* (and can not be queried) and labels that contain "reserved" XML characters.

5. In providing (syntactical) extensions for querying, among others, RDF, Xcerpt becomes a **versatile query language** (as defined in [10]).

6. In a strict separation of querying and construction and in its use of logical variables and deductive rules, it resembles more logic programming languages or Datalog. In contrast, SQL, e.g., mixes construction and querying (nested queries) and uses explicit references to views rather than rule chaining.

These unique characteristics of Xcerpt motivate many of the language concepts introduced in the remainder of this chapter, a more complete discussion of the guiding design principles for (versatile) Web query languages as exemplified in Xcerpt can be found in [10].

3.1 Data Model

As stated above, Xcerpt uses a **graph data model**. More precisely, Xcert provides access to *one or more* data graphs (that are usually stored in data units called "documents" identified by IRIs [21]). Each data graph is a *rooted, directed, node-labeled, ordered, unranked* graph with two types of nodes:

§1 Element Nodes

Element (or structural) nodes represent XML elements or similar **structured** data items (e.g., resources in RDF).

Each element node is decorated further with a dictionary (or associative list) of (XML-style) **attributes**. Some attributes are predefined and exist at all nodes, viz.the label and namespace (cf. [7]), others are specified in the data, e.g., as XML attributes. Just like in XML, attributes are single valued[1] and unordered, i.e., for each attribute name (dictionary key) a single value exists and the order of the key-value pairs is not significant and can not be queried.

Currently, element nodes in Xcept do not have an explicit object or node identity, i.e., two element nodes with the same attributes and children can not be distinguished from each other. Though, explicit node identity can be simulated with the current approach, direct support of explicit node identity is under consideration, cf. Issue 19.

Element nodes closely resemble **element information items** from [16] with two minor deviations: Xcerpt does, at least at the time of writing, not provide access to in-scope namespaces (cf. Issue 27) and for the base URI according to the XML Base recommendation [31] (cf. Issue 26). The handling of attributes, however, deviates notably from the XML information set to emphasize the distinction of elements and attributes: attributes are simple key-value pair, where the key is an XML name (and thus may consist in prefix and local name) and the value is an arbitrary string. No further information can be attached to attributes.

Each element nodes has zero or more edges to other nodes, called its **children**. These edges are always *ordered*. However, in contrast to pure XML, one can specify whether this order is significant, i.e., whether it has to be preserved during storage or transformation and can be queried. All element nodes originating from XML documents are by default ordered (cf. Section 4.6). Element nodes where the order is significant are called *ordered*, element nodes where the order is insignificant *unordered*. There are no further restrictions on the edges, i.e., the graph may be cyclic, may have loops, the same two nodes may be connected by several nodes, e.g., if a node is the 2nd, 4th, and 12th children of another one.

§2 Content Nodes

Content (or atomic) nodes represent data items that are considered **unstructured** in the context of Xcerpt.

Content nodes can be further distinguished into

1. **Text nodes** that represent the textual content of element nodes. The only attribute of

[1] See Issue 18 for a discussion on list-valued attributes (such as attributes of type IDREFS in XML).

a text node is the string it represents. The same restrictions as for text nodes in XSLT [13], XQuery, and XPath [23] apply, i.e., (1) text nodes *never* represent an *empty string*, (2) two text nodes can *never be direct siblings* of each other. Two nodes are direct siblings, if either they are children of the same ordered element node and are consecutive in the children order or they are children of the same unordered element node. Thus, an unordered element node may not have more than one text node child (cf. Issue 21). If two text nodes are constructed as direct siblings they are *collapsed*.

2. **Comment nodes** that represent comments, i.e., annotations on the actual data that are not meant for machine processing. As text nodes, they have only one attribute: the content of the comment. However, in contrast to text nodes no further restrictions are placed on comment nodes.

3. **Processing instruction nodes** that represent processing instructions, i.e., annotations on the actual data that are meant for processing by specific "target" services. They carry two attributes, the content of the processing instruction (usually some form of instructions for the "target" service) and the name of the "target" service.

3.1.1 Terms for Representing Data and Queries

Inspired by logic programming languages, Xcerpt chooses the concept of **terms** for representing, constructing, and querying complex (or structured) data: Xcerpt uses three forms of terms:

1. **data terms** for representing semi-structured data, i.e., all node types from the data model are represented as terms,

2. **construct terms** for specifying "forms" or "templates" of data to be constructed with variables to indicate where data obtained from the (separated) query part is to be "filled" in, and finally

3. **query terms** for specifying "patterns" or "examples" of the data to be matched by a query again with variables to indicate where data is to be extracted from the matches.

(XML) element nodes represented as terms are the **only complex data structure** in Xcerpt. In particular, variables can only be of type term (which includes literal content such as strings as atoms). Other complex data structure such as lists (or sequences), homogeneous or heterogeneous records, sets, and dictionaries (or associative lists) can be simulated as terms, but no specific constructs are offered. Instead Xcerpt avoids to burden the query author with the selection of the appropriate data structures and leaves this to the query processor. The query processor can choose appropriate storage and access methods, if a term is restricted, e.g., by means of a schema (i.e., type information, see Section 8.3). E.g., a term's children may be stored and accessed using algorithms for dictionaries if it is known that the labels of all children are mutually distinct. Or duplicate elimination may be skipped during query evaluation if the children of a term are indeed restricted to a proper set.

The remainder of this chapter introduces the three concrete syntaxes for Xcerpt discussed in this article: the textual non-XML term syntax, the pure XML syntax, and the hybrid XML-style term syntax.

3.2 A Textual Non-XML Term Syntax for Xcerpt

Xcerpt exhibits not just one concrete syntax, but rather three, each focused on providing a unique set of strengths. The first, non-XML syntax is based around the idea of representing **terms as in logic-programming** and the following principles:

1. Terms are represented **similar to logic-programming**: prefix notation with bracketed argument lists for the children of the term. Special provisions are made to adapt this basic principle to handle the specificities of XML and other Web formats.
2. Different types of brackets encode term properties and distinguish language constructs from data.
3. The syntax strives to be *concise*, but still *easy to read*. The latter objective is supported, e.g., by the preference for explicit full-word keywords (e.g., **`optional`**) to represent language constructs instead of shorthand notations (such as `@` for attribute in XPath).
4. The non-XML term syntax emphasizes that Xcerpts data model and language features are not specific for *one* representation formalism such as XML and RDF, but rather allow different ones to be handled with the same concepts by mapping them to Xcerpt terms, still providing for all the specificities of the supported representation formalisms.

The actual syntax is introduced in each chapter along the abstract and the other concrete syntaxes. The following preliminary remarks set the basis for the discussion of the non-XML term syntax in the rest of this article.

3.2.1 Lexical Structures

The textual non-XML term syntax makes use of the following five lexical structures:

1. **Names:** For, among others, element labels and variables, Xcerpt uses the namespace-aware identifiers, that must adhere to the definition for ⟨*NCName*⟩ in the W3C XML Namespace recommendation [7]. Notice, that this allows for slightly different identifiers than allowed by the definition of an XML (1.0) ⟨*Name*⟩ in [8]. The difference is that in namespace-aware identifiers as used in Xcerpt the double colon characters is *not* allowed.
2. **IRIs:** For namespaces and as a pool for unique identifiers, Internationalized Resource Identifiers (short IRIs) may be used in Xcerpt. Internationalized Resource Identifiers are defined in RFC 3987 [21]. Like strings, IRIs are always enclosed in straight double quotes in Xcerpt.
3. **Strings:** Literal content is represented as strings of characters. However, to avoid the introduction of character entities into Xcerpts non-XML syntax, Java strings (as of §3.10.5 of [25]) are chosen and not, e.g., XML character data. Since Xcerpt's syntax is not line-oriented, there is no need to escape linefeed or carriage return. Thus, an Xcerpt string is an arbitrary sequence of Unicode characters with straight double quotes and backslashes backslash-escaped. For convenience, Java escape sequence (e.g., `\t` for a tabulator) and Unicode escapes (e.g., `\u000d` for a carriage return) are also allowed. Strings in Xcerpt are *always* enclosed in straight double quotes ("), never in single straight quotes.
4. **Numbers:** Some Xcerpt constructs have parameters that are natural numbers. Here, we use again the definition from [8].
5. **Regular Expressions:** In query terms (cf. Chapter 6), Xcerpt uses POSIX.1 regular expressions as defined in [26], Base Definitions Volume (XBD), ch. 9, but extends these regular expressions with variable bindings.

This results in the following grammar for the lexical structures used in Xcerpt's non-XML term syntax (lexical structures are distinguished from other non-terminals by an uppercase first letter):

⟨*NCName*⟩ ::= ⟨*http://www.w3.org/TR/REC-xml-names/#NCName*⟩

⟨*IRI*⟩ ::= '"' ⟨*http://www.ietf.org/rfc/rfc3987.txt#IRI*⟩ '"'

⟨*String*⟩ ::= '"' ⟨*StringCharacter*⟩* '"'

⟨*StringCharacter*⟩ ::= ⟨*http://java.sun.com/docs/books/jls#StringCharacter*⟩ | ⟨*Line-feed*⟩ | ⟨*Carriage-return*⟩

⟨*Line-feed*⟩ ::= '\u000a'

⟨*Carriage-Return*⟩ ::= '\u000d'

⟨*Int*⟩ ::= ⟨*http://www.w3.org/TR/REC-xml-names/#Digit*⟩*

⟨*Regexp*⟩ ::= '/' ⟨*{http://www.unix.org/version3/ieee_std.html#}extended_reg_exp*⟩ '/'

⟨*http://www.unix.org/version3/ieee_std.html#ERE_expression*⟩ ::=
⟨*http://www.unix.org/version3/ieee_std.html#one_char_or_coll_elem_ERE*⟩
| '^'
| '$'
| '(' ⟨*http://www.unix.org/version3/ieee_std.html#extended_reg_exp*⟩ ')'
| '(' ⟨*variable*⟩ '->'
⟨*http://www.unix.org/version3/ieee_std.html#extended_reg_exp*⟩')'
| ⟨*http://www.unix.org/version3/ieee_std.html#ERE_expression*⟩
⟨*{http://www.unix.org/version3/ieee_std.html#}ERE_dupl_symbol*⟩

Notice, how the ⟨*ERE_expression*⟩ from the POSIX standard is overwritten by the production specified here that includes (in line 6 of the production) the added syntax for binding Xcerpt variables in regular expressions. For ⟨*variable*⟩ production is defined in Chapter 6.

3.2.2 Reserved Names

Xcerpt's non-XML term syntax does not reserve any names as language keywords, as the structured of the language allows a disambiguation between keywords and names.

However, the Xcerpt namespace identified by the IRI `http://xcerpt.org/ns/core/1.0` is reserved for language constructs and can not be used for other purposes.

Some implementations may additionally want to restrict the occurrence of keywords in identifiers, i.e., the may want to chose the following restricted definition of ⟨*NCName*⟩:

⟨*NCName*⟩ ::= ⟨*http://www.w3.org/TR/REC-xml-names/#NCName*⟩ - ⟨*ReservedNames*⟩

⟨*ReservedNames*⟩ ::= 'all' | 'and' | 'desc' | 'descendant' | 'except' | 'first' | 'not' | 'optional' | 'or' | 'position' | 'some' | 'without'

3.2.3 Whitespace and Comments

In the grammars for the non-XML term syntax, whitespace is not explicitly included. Rather Xcerpt uses the following whitespace handling rules (similar to, e.g., XQuery [6]):

1. Arbitrary sequence of whitespace characters (as defined by the character class ⟨*S*⟩ in the XML specification [8]) and Xcerpt comments may occur in between any two terminals and must occur where otherwise two sequential terminals are recognized as one. It can be safely "normalized" to a single whitespace character.
2. Strings, names, and other literal structures are considered a single terminal for the purpose of this rule. In other words, in strings, names, and other literals whitespace is significant and may not be ignored. E.g., the string " a " differs from the string "a".

Xcerpts non-XML term syntax allows both end-of-line and block **comments** to occur in place of whitespace. The following rules define whitespace and comments for Xcerpt's non-XML term syntax.

⟨*Whitespace*⟩ ::= (⟨*http://www.w3.org/TR/REC-xml/#S*⟩ | ⟨*End-of-line-comment*⟩ | ⟨*Block-comment*⟩)*

⟨*Comment-char*⟩ ::= ⟨*http://www.w3.org/TR/REC-xml/#NT-Char*⟩

⟨*End-of-line*⟩ ::= ⟨*Line-feed*⟩ | ⟨*Carriage-return*⟩ (⟨*Line-feed*⟩)?

⟨*End-of-line-comment*⟩ ::= '#' (⟨*Comment-char*⟩* – (⟨*Comment-char*⟩* ⟨*End-of-line*⟩ ⟨*Comment-char*⟩*) ⟨*End-of-line*⟩

⟨*Block-comment*⟩ ::= '/#' (⟨*Comment-char*⟩* – (⟨*Comment-char*⟩* ('/#' | '/#') ⟨*Comment-char*⟩*) '/#'

Notice, that block comments can not be nested, cf. Issue

3.3 Hybrid XML-style Term Syntax

The hybrid XML-style term syntax is a rather recent development. It aims at

1. providing a syntax that is (almost) **immediately accessible** to persons accustomed with XML;
2. **very explicit**, i.e., uses in addition to the XML syntax uses only English words to represent language features;

This makes the syntax both easier to read and harder to write, as it is slightly less compact than the term syntax but therefore often uses English words to represent language features instead of special symbols.

The mixing of XML-style syntax for terms and keywords as in the non-XML syntax raises a number of potential clashes. Most notably, *character data still has to be quoted* in contrast to XML to avoid having to escape non-XML parts of the syntax.

As the remainder of this article illustrates, that the hybrid XML-style term syntax can indeed be defined by very few deviations from the non-XML term syntax that only affect the representation of (structured) terms as XML-style elements instead of logic-programming style.

The same *lexical structures* and *reserved words* as in the non-XML syntax are used. In particular, character data must be quoted as in the non-XML syntax. Neither normal XML character data nor CDATA sections are allowed.

Note, that this is possible as the Xcerpt term syntax follows the same convention as XML for plain names.

The same syntax for end-of-line and block comments as in Xcerpt's term syntax is used. Note, that block comments in the XML syntax, i.e., using <!-- and --> as delimiters, may also occur, but represent comments in the *data*, not in the query language (cf.Section 4.2).

3.4 Pure XML Syntax

The previous two syntaxes are mainly intended for human use. Like Relax NG, Xcerpt also exhibits a pure XML syntax that, though harder to author and read for humans, is easy to process with XML tools. The guiding principle is a form of *markup reification*, i.e., elements and attributes are explicitly represented by XML elements named element and attribute (similar to XSLT's xsl:element and **`xsl:attribute`** elements for construction of elements with computed names).

The *lexical structures* and *whitespace handling* rules used are, of course, those of XML, see [8]. In particular, character data must follow the XML restrictions, i.e., < and & must be escaped.

As in the other concrete syntaxes, the Xcerpt namespace `http://xcerpt.org/ns/core/1.0` is reserved to indicate language constructs. Indeed, in the XML syntax *all* language constructs are expressed through elements in the Xcerpt namespace.

Chapter 4

Specifying Semi-structured Data: Xcerpt's Data Terms

4.1 Defining Data Terms

Starting with this section, the remainder of this chapter discusses the three different kinds of terms used in Xcerpt starting with data terms, the most basic term kind.

§1 Data Terms

Data terms are (linear) representations of semi-structured *data* in Xcerpt.

Unsurprisingly, data terms are closely aligned with Xcerpt's data model as introduced in Section 3.1. Each of the node types in Xcerpt's data model are represented by a corresponding kind of data term. However, data terms differ in two notable aspects:

First, data terms are hierarchical (i.e., tree shaped). Thus, to obtain a linear representations of the Xcerpt data graph, **referable term identifiers** and **references** are introduced that allow to express non-hierarchical relations.

§2 Term Identifiers

Referable term identifiers are identifiers for (structured) terms (representing element nodes) that are *unique* at least within the current document and allow *references* to the identified terms.

Term identifiers are only required to be unique within the current document (cf. Issue 20). This allows for easier authoring and validation but excludes out- or cross-document links. Such links must be explicitly traversed using a value join, e.g., in the case of (X)HTML between the fragment identifier in an `href` attribute and the `id` attributes in the target document.

§3 References

References are "links" in the linear term syntax for representing non-hierarchical relations. They are *transparently* resolved, i.e., the case of a term containing a reference to another term can not be distinguished from the case where the term contains the other term as a direct child. For all references in a document, there must be *exactly one* term with a term identifier identical to the value of the reference in the *same* document.

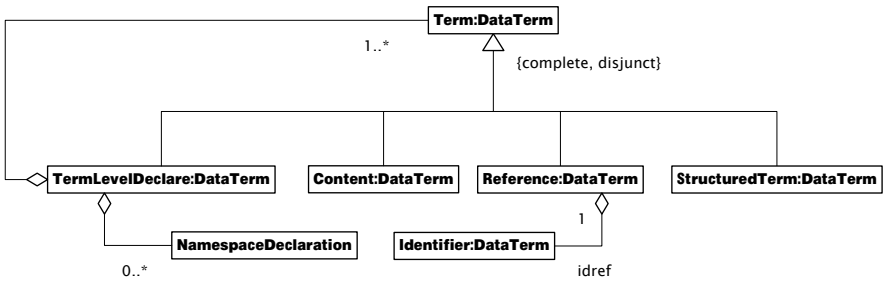


Figure 4.1: UML Model of Data Terms

At the time of writing, term identifiers and references are exclusively part of the linear term representations and *not* preserved in the data model (cf. Issue 19 on introducing explicit node identity in Xcerpt). As a consequence, *term identifiers can not be queried*.

Second, **namespace declarations** (and thus the concept of in-scope namespaces) are not considered as part of term specifications, but are declared in **declaration blocks** surrounding the terms where they may be used. This allows slightly more flexibility in the block structure for declarations and a consistent treatment of declarations on term and rule level (cf. Section 7). Declaration blocks are also used for other declarations (e.g., variable declarations in query and construct terms or type declarations), but at the time of writing namespace declarations are the only kind of declarations allowed in data term declaration blocks. A more detail description of declarations and declaration blocks is given in Section 7.

Figure 4.1 summarizes the kind of terms found as data terms in form of an UML model (cf. Section: A data term can be either

1. an **atomic or content** data term that represents a content node in the data model from Section 3.1,
2. a **structured** data term that represents an element node in the data model,
3. a **reference** to another (structured) data term expressed by a term identifier, or
4. a term-level **namespace declaration** surrounding any number of other data terms.

4.1.1 Textual Term Syntax: Basic Data Terms

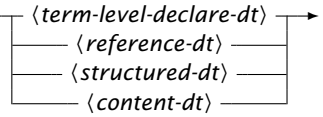
Data terms are defined in the textual term syntax just like in the abstract syntax: either term-level declarations, structured or content data terms, or a reference to another term.

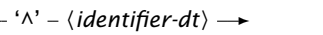
The following productions define first ⟨*data-term*⟩, a basic data-term, and then references (indicated by a ^ symbolizing an ↑ often used to indicate references) and term-level declarations (indicated by the **declare** keyword). Content and structured data terms are discussed in the next sections.

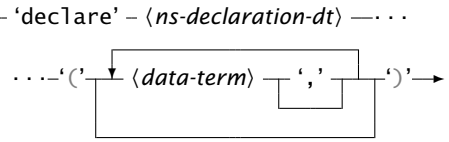
Notice, how the concrete syntax allows both the list of namespace declarations and the data terms in the scope of the declaration to be empty. The abstract syntax (cf.Figure 4.1) however demands that both lists are at least size 1. This is no contradiction: the concrete syntax is designed to be **open**, i.e., to *allow also constructs that are superfluous but not harmful* to ease, e.g., query refactoring and iterative query authoring. In the abstract syntax that presents the information model of an Xcerpt program these superfluous constructs are, however, not any more present. If in the concrete case of term-level

declarations either list is empty, there either have been no namespaces declared or there are no affected data-terms. In both cases, the declaration is ineffectual and will be dropped.

Xcerpt has a considerable number of *parameterized concepts*, i.e., concepts that occur in different contexts with different parameters. E.g., declaration blocks may enclose data terms, top-level data terms, query terms, construct terms, and rules as body, but in each place where a declaration block may occur only one such enclosed construct type is allowed. This form of parameterized concepts can not be directly expressed in notation such as EBNF. Therefore non-terminals that indicate by a suffix the context in which, e.g., a declaration block occurs are used (e.g., ⟨*term-level-declare-dt*⟩ instead of just ⟨*term-level-declare*⟩).

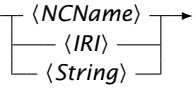
⟨*data-term*⟩ ::= 

⟨*reference-dt*⟩ ::= 

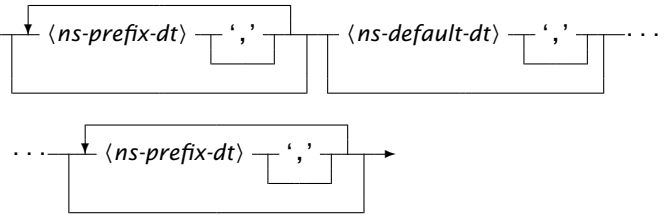
⟨*term-level-declare-dt*⟩ ::= 

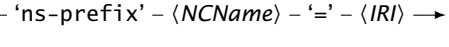
Note, that the parentheses used to enclose the in-scope data terms of the declaration are colored in 'gray'. In the following this is used as a short-hand to indicate that, if a construct that has a list of terms as parameter is applied to (a list of) exactly one, then the brackets can be omitted. I.e., A '(' B* ')' where A and B are arbitrary right-hand side expressions in the EBNF syntax used in this article, is equivalent to (A B) | A '(' B+ ')'.

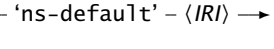
Identifiers (in data terms) are introduced here, but also use in several other parts of a data term, cf.Section 4.3. Identifiers in the non-XML term syntax can be either namespace-aware XML names, IRIs, or strings. As discussed in Section 3.2.1, both IRIs and strings are always enclosed in straight double quotes, only namespace-aware XML names remain unquoted.

⟨*identifier-dt*⟩ ::= 

Namespace Declarations (in data terms) are the basic form of namespace declarations. In query and construct terms namespace declarations are slightly extended to also allow variables instead of prefixes or namespace URIs.

⟨*ns-declaration-dt*⟩ ::= 

⟨*ns-prefix-dt*⟩ ::= 

⟨*ns-default-dt*⟩ ::= 

4.1.2 XML-style Term Syntax: Basic Data Terms

Here, the XML-style term syntax uses the exact same productions as the non-XML term syntax, differences occur only in later parts of data-terms, viz. when defining structured data terms.

4.1.3 Pure XML Syntax: Basic Data Terms

The pure XML syntax is, as mentioned above, defined using Relax NG schemata. To highlight the similarities between data, construct, and query terms and to ensure consistency, all three terms are defined based a common grammar for terms, that is parameterized where needed. In fact, this common grammar exactly captures data terms and is explained in the following.

The first excerpt defines syntax for the basic concepts introduced above: terms, references, and term-level declarations. Syntax for content and structured (data) terms is discussed in the following sections.

```
default namespace = "http://xcerpt.org/ns/core/1.0"

## A generic Xcerpt term. Variants are data, construct, and query terms.
term.class |=
  reference | content-term | structured-term | term-level-declare

## A declaration block on term level allows possibly (in data and construct terms) only namespace declarations.
term-level-declare =
  grammar {
    include "declare-block.rnc" {
      content = parent term.class*
      var-declaration = empty
    }
  }

## The using occurrence of a reference, i.e. "^ id" in term syntax.
reference = element reference { identifier.class }
```

Notice, how term-level declarations are indeed defined by referencing an nested grammar and parameterizing some of its non-terminals, viz. what the content of a declaration is (here a term) and that no variable-declarations are used. Figure 4.2 shows a visualization of the definition of term-level-declare unfolding the nested grammar.

Surprisingly, Relax NG restricts the ability to parameterize grammars to inclusion of grammars in external files, here the file `declare-block.rnc`, whose content is the following Relax NG grammar:

```
default namespace = "http://xcerpt.org/ns/core/1.0"
namespace a = "http://relaxng.org/ns/compatibility/annotations/1.0"

start = declare-block

## A declare block with an empty content and both namespace and variable declarations.
declare-block =
  element declare { (ns-declaration | var-declaration)*, content }
ns-declaration =
  ns-prefix-declaration*,
  element ns-default {
    element iri { iri.class }
  }?,
  ns-prefix-declaration*
ns-prefix-declaration =
  element ns-prefix {
    element name { ncname.class },
    element iri { iri.class }
```

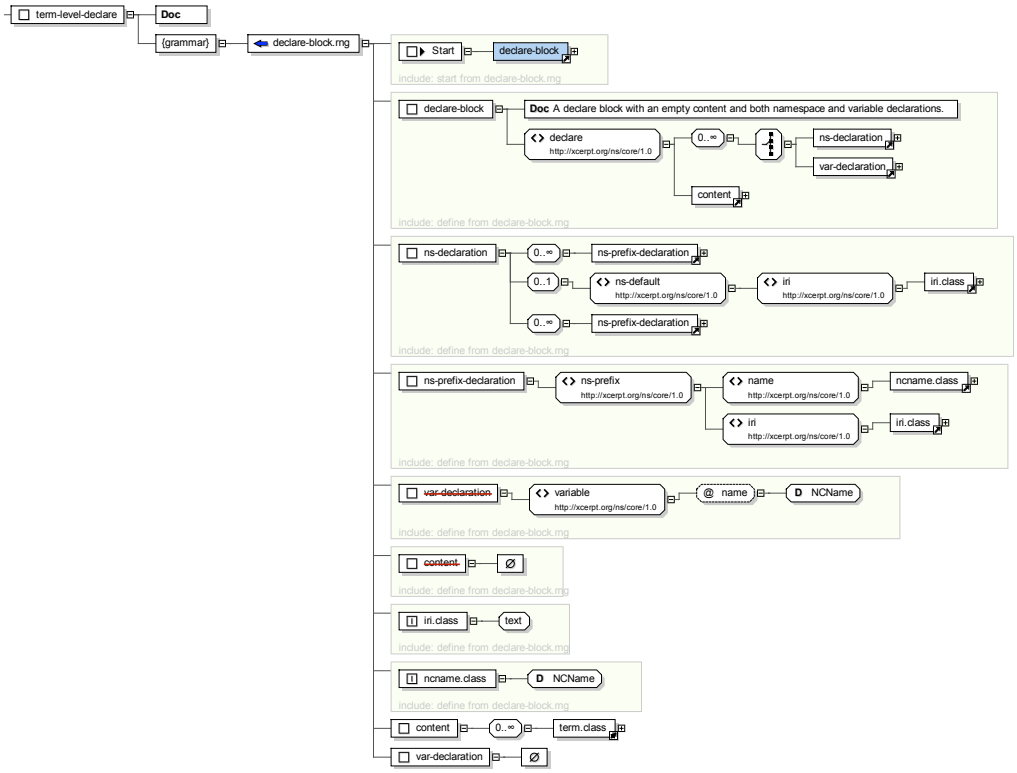


Figure 4.2: Relax NG Schema for Term-level Declarations in Data Terms

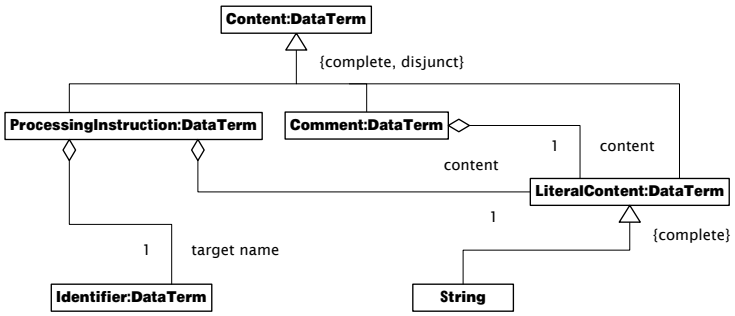


Figure 4.3: UML Model of Atomic Data Terms

```
 }
var-declaration =
 element variable {
  attribute name { xsd:NCName }
 }
content = empty
iri.class |= text
ncname.class |= xsd:NCName
```

Notice, the use of combinable definitions (indicated by |=) for uri.class and ncname.class. This allows later the easy addition of more choices (viz. variables that can occur instead of the plain IRIs or names).

4.2 Content Data Terms

§4 Content Data Terms

The atomic form of data terms are terms that represent information that is considered *unstructured* in the context of Xcerpt, viz. literal (character) content as well as data annotations in form of comments for human consumption and in form of processing instructions for machine consumption.

A formal model of content data terms is shown in Figure 4.3: for each of the content nodes in the data model introduced in Section 3.1 a corresponding data term exists. Notice, that in alignment with [8] neither processing instructions, comments, or literal (character) content can be nested.

XML restricts the literal content of comments (processing instructions, resp.) to disallow the character sequence '–' ('?>', resp.). This is not the case in Xcerpt. However, when creating XML data these additional restrictions have to be considered, cf.Issue 24).

4.2.1 Textual Term Syntax: Content Data Terms

Literal content is represented in the non-XML term syntax by simple (Java-style Unicode) strings as defined in Section 3.2.1. Comments and processing-instructions use the same syntax as structured data terms introduced in the next section: label in prefix position followed by the list of attributes and children. Here, the first is enclosed in (round) parentheses, the second in (square) brackets. Structured

terms may also use curly braces around the children list to indicate that the order is insignificant.

Since comments may have no attributes their attribute list is always empty.

⟨*content-dt*⟩ ::= ⟨*literal-content-dt*⟩ | ⟨*comment-dt*⟩ | ⟨*processing-instruction-dt*⟩

⟨*literal-content-dt*⟩ ::= ⟨*String*⟩

⟨*comment-dt*⟩ ::= 'xcerpt' – ':' – 'comment' – '(' -')- ' ' [' — ⟨*literal-content-dt*⟩ — ']'

⟨*processing-instruction-dt*⟩ ::= 'xcerpt' – ':' – 'processing-instruction' | 'xcerpt' – ':' – 'pi' — '(' – 'target-name' – '=' – ⟨*identifier-dt*⟩ – ')' — '[' — ⟨*literal-content-dt*⟩ — ']'

4.2.2 XML-style Term Syntax: Content Data Terms

Again the XML-style term syntax uses the same productions as the non-XML term syntax, but differs on the representation of comments and processing instructions: Both are represented as in XML with the exception of their literal content. That is, as literal content of elements, quoted by straight double quotes (just like in the non-XML term syntax). Thus if an element in an XML document contains the character sequence `a &amp; b &lt; c \ d or &#x1000;` this may be written as `"a & b < c \\ d or \u1000"` if it occurs in a Xcerpt data term in XML-style term syntax.

⟨*comment-dt*⟩ ::= '<!-' — ⟨*literal-content-dt*⟩ — '->'

⟨*processing-instruction-dt*⟩ ::= '<?' – ⟨*identifier-dt*⟩ — ⟨*literal-content-dt*⟩ — '?>'

Also note, that XML disallows comments containing character sequence '-' (and no quoting mechanism is applicable, as entities are not expanded in comments) as well as processing instructions containing the character sequence '?>' (again no quoting mechanism applicable). These restrictions are not present in Xcerpt or one of its syntaxes.

4.2.3 Pure XML Syntax: Content Data Terms

Again, we show the case for representing general terms in the pure XML syntax:

```
## A content term represents literal or other non-nestable content.
content-term = literal-content.class | annotation-content

## Content kinds that can be used to annotate elements.
annotation-content =
  element comment { literal-content.class }
  | element processing-instruction {
      attribute target { identifier.class },
      literal-content.class
    }

## Character data or other atomic content.
literal-content.class |= text
```

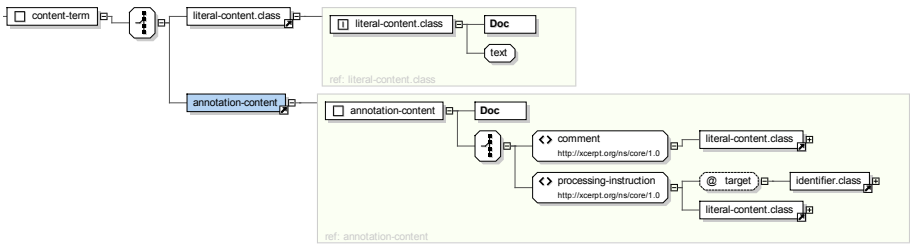


Figure 4.4: Relax NG Schema for Content Terms

Notice, how literal-content.class is "open" (indicated by |=), i.e., further choices for literal-content.class may be added separately. Figure 4.4 shows a visual representation of the Relax NG schema fragment with definitions inlined.

4.3 Structured Data Terms

§5 Structured Data Terms

Structured data terms correspond with element or structural nodes in the data model and can be nested.

Structured data terms can be distinguished in *ordered* and *unordered* data terms. As for element nodes, the distinction indicates

§6 Ordered and Unordered Terms

A (structured) term is called *ordered*, if the order of its children is significant. Otherwise it is called *unordered*. In the former case the order must be preserved during processing and storage and is accessible in queries, whereas in the latter it may change during storage or processing and can not be queried.

For consistency with query terms, structured data terms are further classified as *total*.

§7 Total Terms

A term is called *total*, if its list of children is complete, i.e., there can be no additional children.

In Xcerpt, all data terms are total, i.e., must have all their children specified (cf. Issue 35).

Each structured term consists in two parts: a specification of its attributes (called in the following "local" specification as attributes are the non-structural properties of that term) and its children as shown in Figure 4.5:

- The **children** of a structured data term is a sequence of zero or more (arbitrary) data terms. If the children list is empty, the term itself is often referred to as an *empty* term. This children list corresponds to the child edges in the data model after reference resolution.

- The **local specification** of a structured data term is depicted in Figure 4.6. It allows the specification of

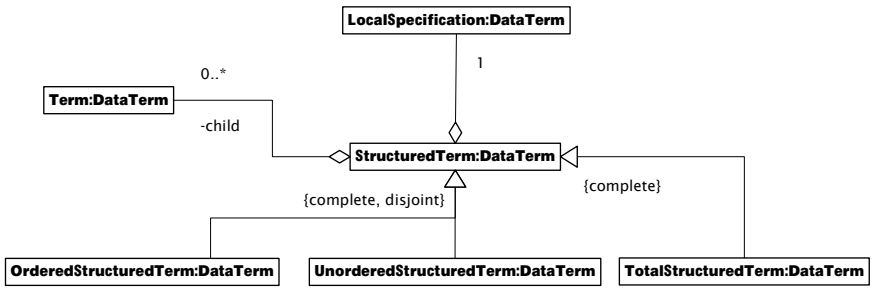


Figure 4.5: UML Model of Structured Data Terms

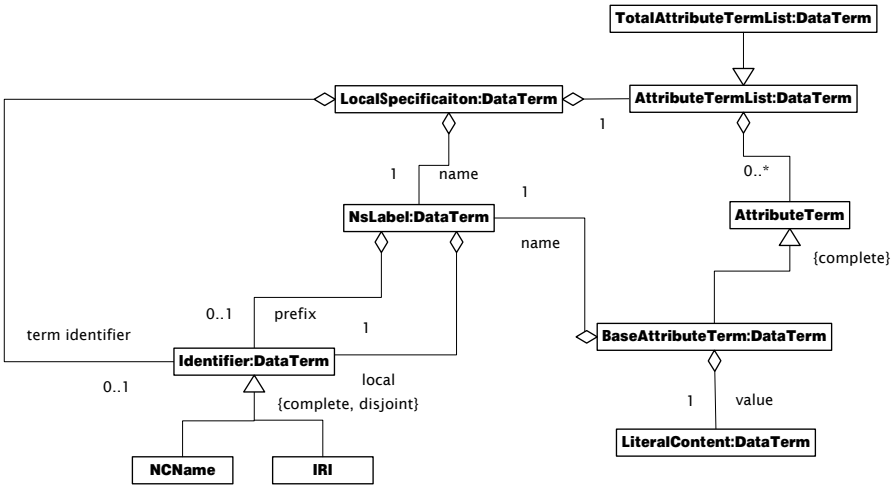


Figure 4.6: UML Model of the Local Specification for Structured Data Terms

1. An optional *term identifier* that can be used to reference the term, as discussed above. Term identifiers can be either XML Names[1] or IRIs.

2. Exactly one name or *label* of the data term. The label itself consists in an optional namespace prefix and a mandatory local part. Again both can be either XML Names or IRIs.

3. An *attribute term list*, that is (a) in data terms always total and (b) consists of one or more attribute terms, which in data terms are simple pairs of attribute names (or key) and values. The name has the same shape as data term labels, the value as literal content data terms.

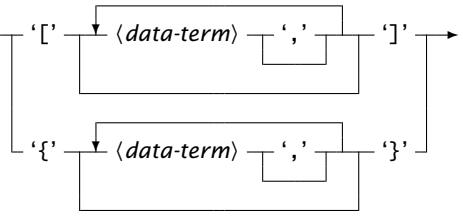
[1] Or, more precisely, NCNames as defined in the W3C XML Namespace recommendation [7]. Notice, that these are not quite identical to the original XML (1.0) Names defined in [8], that do *not* treat the double colon as a special character.

4.3.1 Textual Term Syntax: Structured Data Terms

Structured data terms form the core of the non-XML term syntax: the "local", non-structural properties are specified in prefix notation followed by the list of children. The list of children is enclosed in brackets, either square brackets ('[ ]') to indicate that the order in which the children are specified is significant or curly braces ('{ }') to indicate that the order is insignificant:

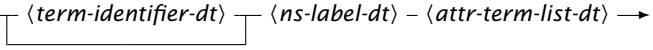
⟨*structured-dt*⟩ ::= ⟨*local-spec-dt*⟩ – ⟨*children-list-dt*⟩

⟨*children-list-dt*⟩ ::=



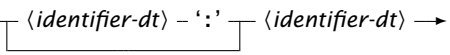
Local properties of a structured term are the optional term identifier (that is preceding the remaining properties and separated from them by '@'), the label of the term, and its attributes. The label itself falls into two parts separated by ':', viz. the optional namespace and the local name. Term identifier, namespace, and local name are all identifiers as defined above in Section 4.1.

⟨*local-spec-dt*⟩ ::=



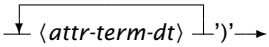
⟨*term-identifier-dt*⟩ ::= ⟨*identifier-dt*⟩ – '@'

⟨*ns-label-dt*⟩ ::=



Attribute terms are specified in a attribute list term, enclosed in round parentheses. Each attribute is a pair, separated by '=', of the attribute label (itself, as an element label, a pair of namespace and local name separated by ':') and the attribute content.

⟨*attr-term-list-dt*⟩ ::=



⟨*attr-term-dt*⟩ ::= ⟨*base-attr-term-dt*⟩

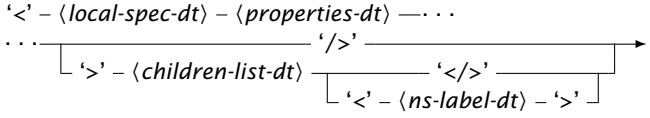
⟨*base-attr-term-dt*⟩ ::= ⟨*ns-label-dt*⟩ – '=' – ⟨*literal-content-dt*⟩

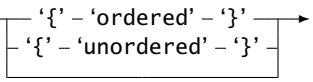
Notice, that both the list of children nor the attribute may be empty, but neither may be absent. I.e., neither `a[]` nor `a()` or `a` are structured terms, but, e.g., `a()[]` and `a(){}`, cf. Issue 3.

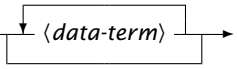
4.3.2 XML-style Term Syntax: Structured Data Terms

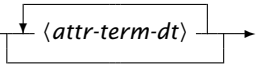
In the representation of structured terms lies the main difference between the non-XML and the XML-style term syntax: instead of prefix notation for terms as in logic-programming, XML-style element and attribute notation is used. To achieve this, the following four productions are changed or added:

⟨*structured-dt*⟩ ::=



⟨*properties-dt*⟩ ::= 

⟨*children-list-dt*⟩ ::= 

⟨*attr-term-list-dt*⟩ ::= 

There is an additional restriction on the first production: the (namespace, local name) pair used as label in the end element tag and the (namespace, local name) pair used in the start element tag (i.e., produced as part of ⟨*local-spec-dt*⟩) must be (modulo whitespace) component wise equivalent character sequences.

Observe, how the first production encloses the entire local spec (including, e.g., the term identifier) in the start element tag. Just like in XML this makes all the attributes part of the start element tag. Instead of using different brackets, the significance of the order is indicated here using special term properties '{ordered}' and '{unordered}'. If neither is given the term is assumed to be ordered as in standard XML.

4.3.3 Pure XML Syntax: Structured Data Terms

Structured terms (and as such structured data terms) are represented in the pure XML syntax by an XML element element with sub-elements for its identifier (optional), its label, its list of children, and its list of attributes.

E.g., the following data term in non-XML term syntax

```
a(b = "c & a \\ b"){ o1 @ d()[ e()[ ] ], d()[ ] }
```

is represented in pure XML syntax as:

```
<element> <!-- a(b = "c & a \\ b"){ o1 @ d()[ e()[ ] ], d()[ ] } -->
  <label>a</label>
  <attributes>
    <attribute total="true">
      <label>b</label>
      <value>c &amp; a \ b</value>
    </attribute>
  </attributes>
  <children ordered="false" total="true">

    <element> <!-- o1 @ d()[ e()[ ] ] -->
      <identifier>o1</identifier>
      <label>d</label>
      <attributes total="true" />
      <children ordered="true" total="true">
        <element>
          <label>e</label>
          <attributes total="true" />
          <children ordered="true" total="true" />
        </element>
      </children>
    </element>

```

```
      <element> <!-- d()[] -->
        <label>e</label>
        <attributes total="true" />
        <children ordered="true" total="true" />
      </element>
    </children>
  </element>
```

Obviously, this is vastly more verbose than either the non-XML or the XML-style term syntax. However, it has the virtue that (with the exceptions of regular and qualified descendant expressions, cf. Chapter 6) all constructs of Xcerpt are explicitly represented as either XML elements or attributes. No non-XML "sub-languages" remain that require special consideration, such as XPath in XSLT. This makes the syntax very easy to process with XML tools.

The following gives the full grammar for structured terms in Relax NG compact syntax:

```
## A structured term is a term that may have children and
## attributes. It contrasts with literal content.
structured-term =
  element element { term-local-spec, term-children, term-condition? }

## Some terms may have additional constraints attached to them.
term-condition = empty

## The children of a term can be ordered or unordered, total or partial.
term-children =
  element children {
    attribute ordered { "true" | "false" },
    attribute total { total.class },
    term.class*
  }

## The specification of the 'local' properties of a term: identifier, label, namespace, and attributes.
term-local-spec = term-identifier?, ns-label, attr-term-list

## The defining occurrence of a reference, i.e. "id @" in term syntax.
term-identifier = element identifier { identifier.class }

## Label and namespace of an Xcerpt term or attribute.
ns-label =
  element label {
    element ns { identifier.class }?,
    identifier.class
  }

## A term specifying the attributes of an element.
attr-term-list =
  element attributes {
    attribute total { total.class },
    attribute-term.class*
  }

## Class of values for attributes specifying totality or
## partiality of a term's children or attribute list.
total.class |= "true"

## A attribute term is an attribute possibly modified with respect to location, modality, and selection.
attribute-term.class |= base-attribute

## An attribute consists of a label and an attribute content.
```

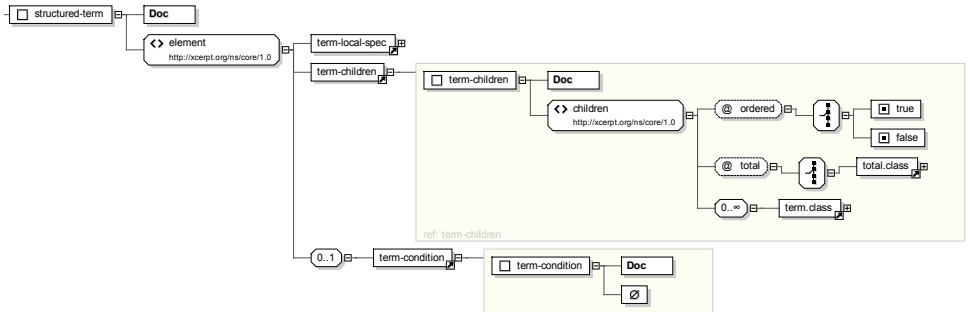


Figure 4.7: Relax NG Schema for Structured Terms

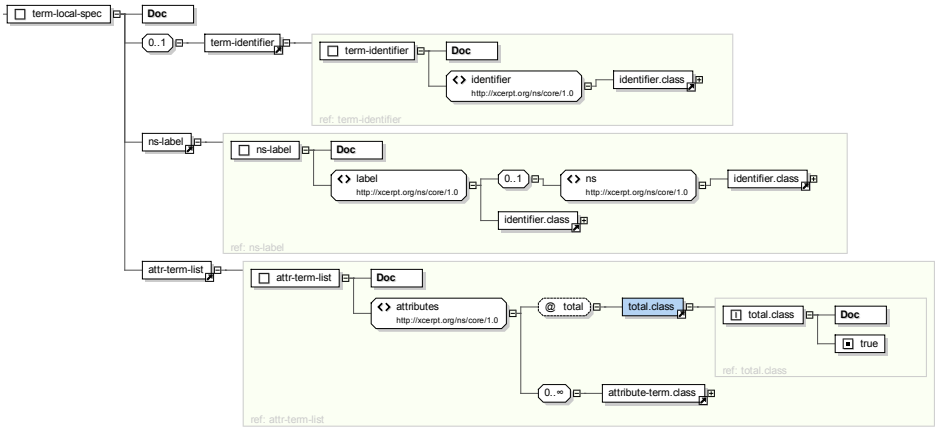


Figure 4.8: Relax NG Schema for Local Specifications of Structured Terms

```
base-attribute =
  element attribute {
    ns-label,
    element value { literal-content.class }
  }
```

In the grammar (and, more easily recognizable, in Figure 4.7 where the productions are inlined) term conditions are provided but as an empty production. This is a sort of "hook" where in query terms actual term conditions can be plugged in. Similarly, the attribute total (cf. also Figure 4.8 showing the local part of a term specification) might strike as peculiar, since it is fixed to the value 'true'. However, the possible values are defined in the non-terminal total.class for which here only a single production is given, but others might be added, e.g., when defining query terms leading to a choice of attributes.

4.4 Top-level Data Terms

To conclude the discussion of data terms, it should be noted, that data terms on top-level are slightly restricted in comparison to data terms at any other level as discussed so far: Only structured data terms and declare blocks are allowed, the later again being restricted to contain

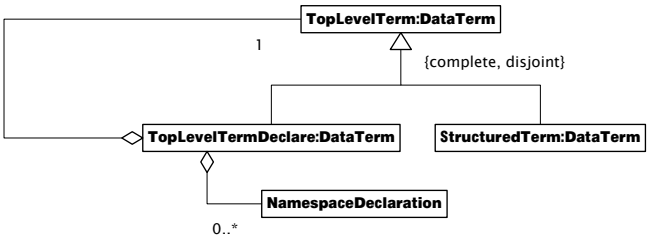


Figure 4.9: UML Model of Top Level Data Terms

only a *single* top-level data term instead of one or more arbitrary data terms. Figure 4.9 shows the full model of top-level data terms.

4.4.1 Textual Term Syntax: Top-Level Data Terms

Due to the different declaration blocks (i.e., basic data terms may be declaration blocks that contain all forms of basic terms, top-level data terms may be declaration blocks that contain only top-level terms), a top-level data term can not be specified as a basic data term with some exceptions. Rather a separate production is needed:

⟨*top-level-data-term*⟩ ::= ►– ⟨*top-term-level-declare-dt*⟩ - ⟨*structured-dt*⟩ –►

⟨*top-term-level-declare-dt*⟩ ::= ►– ‘declare’ – ⟨*ns-declaration-dt*⟩ – ‘(’ – ⟨*top-level-data-term*⟩ – ‘)’ –►

4.4.2 XML-style Term Syntax: Top-Level Data Terms

Again, the productions for the XML-style term syntax do not differ from the non-XML term syntax.

4.4.3 Pure XML Syntax: Top-Level Data Terms

The treatment of top-level terms concludes the definition of terms (which are, as stated above, exactly data terms). Terms are defined in a separate grammar and will be included in the full Xcerpt syntax three times, once for each type of term. For data terms, no parameterization occurs at inclusion:

```
# Data Terms
data-term =
  grammar {
    include "term.rnc"
  }
```

`term.rnc` contains the grammar for terms including all productions discussed so far and the following fragment for top-level terms:

```
default namespace = "http://xcerpt.org/ns/core/1.0"

start = top-level-term.class

## A term that may occur at top-level. Slightly more
## restricted than a basic term.
top-level-term.class =
  structured-term
  | grammar {
```

```
    include "declare-block.rnc" {
      content = parent top-level-term.class*
      var-declaration = empty
    }
  }
```

Notice, how the start symbol of the grammar is set to top-level-term.class. The above inclusion of the grammar thus makes data-term an alias for top-level-term.class from `term.rnc`.

4.5 Exemplary Data Term

The following data term will be used as running example for the remainder of this article (e.g., as basis for query and construct term examples). It is drawn from the domain of bibliography management: Mixing typical bibliographic records (similar to Bibtex or DBLP) with actual content (represented as XHTML or in a Docbook-style format) it combines

- so-called document-oriented with data-oriented XML, i.e., data with flexible, recursive structure and data with rather rigid and flat structure. Recursive structure is used, e.g., for the content of articles in Docbook-style format.
- normalized with de-normalized representation of data (e.g., author information is duplicated for each authored paper, whereas the information about the journal is represented once and referenced in other parts),
- hierarchical with delimiter-based structuring of data (e.g., (X)HTML style sections delimited by consecutive hn elements vs. nested sections as, e.g., in DocBook),
- resolved and unresolved links. Where links are used to normalize the data (e.g., in case of journal information of a an article), these links are resolved to Xcerpt references. Other links (e.g., the link to another section in the content of an article) are left unresolved as they must be distinguished from "normal" nesting. E.g., links to other sections in the content of an article (like "cf.Section 10") part-of relations).

Figures 4.10 and 4.11 show fictional journal, proceeding, and article information in non-XML and XML-style term syntax. Notice, the close relation of these syntactical variants: each line of the non-XML term syntax has exactly one corresponding line in the XML-style term syntax.

Unsurprisingly, in pure XML syntax the sample data is considerably longer than in either of the term syntaxes (715 lines vs. 130 lines, i.e., a 5.5-fold increase): Essentially, the entire syntax tree is represented explicitly as XML elements. The full sample data is shown in Appendix C.3, an excerpt (the full journal entry) in Figure 4.12.

4.6 XML Documents as Data Terms

Before taking a look at how queries—that is selection of existing data and construction of new data—are expressed in Xcerpt, the discussion of data terms in Xcerpt is concluded by a look at how XML documents are transformed into data terms when queried with Xcerpt.

An XML document contains (leaving aside for the moment prolog and epilog) a single document element that can be accessed in Xcerpt as a single data term. Section 6.4.2 introduces document specifications that also allow access to prolog and epilog of an XML document. Here,

```
bib(){
  journal.adm @ journal(){
    title()[ "Applied Data Management" ]
    editors()[
      editor-in-chief()[ "Titus Pomponius Atticus" ]
      editor(region="Africa")[ "Marcus Aemilius Aemilianus" ]
      editor(region="Gaul")[ "Aulus Hirtius"
        affiliation()[ "Governor, Transalpine Gaul" ] ]
      editor(region="Cilicia")[ "Marcus Tullius Cicero"
        affiliation()[ "Governor, Cilicia" ] ]
    ]
    publisher()[ "Titus Pomponius Atticus" ]
    volumes()[
      journal.adm.v10 @ volume()[
        journal.adm.v10.n1 @ number(type="special-issue"){
          title()[ "Data Processing Challenges in the Age of Wax Tablets" ]
          editorial()[ ^ articles.66.cicero.wax ]
          year()[ "60" ]
          month()[ "july" ]
        }
        journal.adm.v10.n2 @ number(){
          year()[ "60" ]
          month()[ "november" ]
        }
      ]
    ]
  ]
```

```
<bib {unordered}>
  <journal.adm @ journal {unordered}>
    <title>"Applied Data Management"</title>
    <editors>
      <editor-in-chief>"Titus Pomponius Atticus"</editor-in-chief>
      <editor region="Africa">"Marcus Aemilius Aemilianus"</editor>
      <editor region="Gaul">"Aulus Hirtius"
        <affiliation>"Governor, Transalpine Gaul"</affiliation></editor>
      <editor region="Cilicia">"Marcus Tullius Cicero"
        <affiliation>"Governor, Cilicia"</affiliation></editor>
    </editors>
    <publisher>"Titus Pomponius Atticus"</publisher>
    <volumes>
      <journal.adm.v10 @ volume>
        <journal.adm.v10.n1 @ number type="special-issue" {unordered}>
          <title>"Data Processing Challenges in the Age of Wax Tablets"</title>
          <editorial>^ articles.66.cicero.wax</editorial>
          <year>"60"</year>
          <month>"july"</month>
        </number>
        <journal.adm.v10.n2 @ number {unordered}>
          <year>"60"</year>
          <month>"november"</month>
        </number>
      </volume>
    </volumes>
  </journal>
```

```
  conf.dmmc @ proceedings(){
    editors()[
      editor()[ "Marcus Aemilius Lepidus"
        affiliation [ "Consul, SPQR" ] ]
      editor()[ "Gaius Julius Caesar Octavianus" ]
      editor()[ "Marcus Antonius" ]
    ]
    title()[
      "Advancements in Data Management for Military and Civil Application"
    ]
    invited-papers()[
      ^inproc.44.brutus
      ^article.66.scaurus.qumran
    ]
    abbrev()[ "DMMC" ]
    year()[ "44" ]
    month()[ "july" ]
    location()[ "Mutina" ]
    publisher()[ "SPQR" ]
  }
```

```
  <conf.dmmc @ proceedings {unordered}>
    <editors>
      <editor>"Marcus Aemilius Lepidus"
        <affiliation>"Consul, SPQR"</affiliation></editor>
      <editor>"Gaius Julius Caesar Octavianus"</editor>
      <editor>"Marcus Antonius"</editor>
    </editors>
    <title>
      "Advancements in Data Management for Military and Civil Application"
    </title>
    <invited-papers>
      ^inproc.44.brutus
      ^article.66.scaurus.qumran
    </invited-papers>
    <abbrev>"DMMC"</abbrev>
    <year>"44"</year>
    <month>"july"</month>
    <location>"Mutina"</location>
    <publisher>"SPQR"</publisher>
  </proceedings>
```

```
  article.66.scaurus.qumran @ article(){
    author()[ "Marcus Aemilius Scaurus"
      affiliation()[ "Tribun, Gnaeus Pompeius Magnus" ] ]
    title()[ "From Wax Tablets to Papyri: The Qumran Case Study" ]
    in(scrolls="102-112")[ ^ journal.adm.v10.n1 ]
    citations [
      cite(ref="article.66.cicero.wax")[ ]
      cite(type="formatted")[ "M. Aemilius Scaurus (104): A Case for Permanent
          Storage of Senate Proceedings. In: M. Aemilius Scaurus, ed. (104): "
          i()[ "Princeps Senatus: Honor and Responsibility" ]
          ", Chapter 2, 14-88." ]
    ]
  ]
```

```
  <article.66.scaurus.qumran @ article {unordered}>
    <author>"Marcus Aemilius Scaurus"
      <affiliation>"Tribun, Gnaeus Pompeius Magnus"</affiliation></author>
    <title>"From Wax Tablets to Papyri: The Qumran Case Study"</title>
    <in scrolls="102-112">^ journal.adm.v10.n1</in>
    <citations>
      <cite ref="article.66.cicero.wax" />
      <cite type="formatted">"M. Aemilius Scaurus (104): A Case for Permanent
          Storage of Senate Proceedings. In: M. Aemilius Scaurus, ed. (104): "
          <i>"Princeps Senatus: Honor and Responsibility"</i>
          ", Chapter 2, 14-88."</cite>
    </citations>
  </article>
```

Figure 4.10: Exemplary Data Term, Part I: Non-XML and XML-style Term Syntax

```
article.66.cicero.wax @ article(){
  authors()[
    author()[ "Marcus Tullius Cicero"
      affiliation()[ "Governor, Cicilia" ] ]
    author()[ "Marcus Aemilius Lepidus"
      affiliation()[ "Gens Aemilia" ] ]
    author()[ "Marcus Tullius Tiro"
      affiliation()[ "Secretary, M. T. Cicero" ] ]
  ]
  title()[ "Space- and Time-Optimal Data Storage on Wax Tablets" ]
  in(scrolls="1-94")[ ^ journal.adm ]
  content(type="xhtml")[
    declare ns-default "http://www.w3.org/1999/xhtml"
    body()[
      xcerpt:comment[ "incomplete due to melted letters on some tablets" ]
      h1(id="contributions")[ "Contributions" ]
      h1()[ "A History of Data Storage: From Stone to Parchment" ]
      p()[ "Despite " cite()[ ^ article.66.scaurus.qumran ] ... ]
      ol()[
        li()[ em()[ strong()[ "Homeric" ] " Age:" ] ... ]
        li()[ em()[ "Age of the " strong()[ "Kings" ] ":" ] ... ]
      ]
      h1(id="tiro")[ "Notae Tironianae" ]
      img(title="Tironian et" src=...)[ ]
      p()[ "As discussed in " a(href="#contributions")[ ... ] ]
      h1(id="tachygraphy")[ "Challenges for Tachygraphy on Wax" ]
      p()[ "Though conditions for writing on wax tablets are adverse
        to tachygraphy, systems as described in " a(href="#tiro")[ ... ] ]
    ]
  ]
}
```

```
<article.66.cicero.wax @ article {unordered}>
  <authors>
    <author>"Marcus Tullius Cicero"
      <affiliation>"Governor, Cicilia"</affiliation></author>
    <author>"Marcus Aemilius Lepidus"
      <affiliation>"Gens Aemilia"</affiliation></author>
    <author>"Marcus Tullius Tiro"
      <affiliation>"Secretary, M. T. Cicero"</affiliation></author>
  </authors>
  <title>"Space- and Time-Optimal Data Storage on Wax Tablets"</title>
  <in scrolls="1-94">^ journal.adm.v10.n1</in>
  <content type="xhtml">
    declare ns-default "http://www.w3.org/1999/xhtml"
    <body>
      <!-- "incomplete due to melted letters on some tablets" -->
      <h1 id="contributions">"Contributions"</h2>
      <h1>"A History of Data Storage: From Stone to Parchment"</h1>
      <p>"Despite "<cite>^ article.66.scaurus.qumran</cite> ...</p>
      <ol>
        <li><em><strong>"Homeric"</strong>" Age:"</em>...</li>
        <li><em>"Age of the "<strong>"Kings"</strong>":"</em>...</li>
      </ol>
      <h1 id="tiro">"Notae Tironianae"</h1>
      <img title="Tironian et" src=... />
      <p>"As discussed in "<a href="#contributions">...</a></p>
      <h1 id="tachygraphy">"Challenges for Tachygraphy on Wax"</h1>
      <p>"Though conditions for writing on wax tablets are adverse
        to tachygraphy, systems as described in "<a href="#tiro">...</a></p>
    </body>
  </content>
</article>
```

```
inproc.44.brutus @ inproceedings(){
  authors()[
    author()[ "Marcus Antonius"
      affiliation()[ "Consul, SPQR" ] ]
    author()[ "Decimus Junius Brutus"
      affiliation()[ "Governor, Cisalpine Gaul" ] ]
  ]
  title()[ "Efficient Management of Rapidly Changing Personal Records" ]
  in(scrolls="24-48")[ ^ conf.dmmc ]
  content(type="docbook") [
    declare ns-default "http://example.org/ns/docbook/simplified/1.0"
    section()[ info()[ title()[ "Introduction" ] ]
      section()[ info()[ title()[ "Contributions" ] ]
        para()[ "The most notable contributions of this article include:"
          list(type="ordered")[
            item()[
              para()[ "A new " em()[ "methodology" ] " to ..., cf. "
                pageref(idref="inproc.44.brutus.s1")[ ... ] ]
              figure()[ title()[ "Chart of Desertions" ]
                      img()[ ... ] ]
              para()[ "As " cite()[ ^article.66.cicero.wax ] ... ]
            ]
          ]
        ]
      ]
    ]
    inproc.44.brutus.s1 @ section()[
      info()[ title()[ "Acknowledgements" ] ]
      para()[ "We would like to thank the editors of "
        cite()[ ^journal.adm.v10.n1 ] ... ]
    ]
  ]
}
```

```
<inproc.44.brutus @ inproceedings {unordered}>
  <authors>
    <author>"Marcus Antonius"
      <affiliation>"Consul, SPQR"</affiliation></author>
    <author>"Decimus Junius Brutus"
      <affiliation>"Governor, Cisalpine Gaul"</affiliation></author>
  </authors>
  <title>"Efficient Management of Rapidly Changing Personal Records"</title>
  <in scrolls="24-48">^ conf.dmmc</in>
  <content type="docbook">
    declare ns-default "http://example.org/ns/docbook/simplified/1.0"
    <section><info><title>"Introduction"</title></info>
      <section><info><title>"Contributions"</title></info>
        <para>"The most notable contributions of this article include:"
          <list type="ordered">
            <item>
              <para>"A new "<em>"methodology"</em>" to ..., cf. "
                <pageref idref="inproc.44.brutus.s1" /> ...</para>
              <figure><title>"Chart of Desertions"</title>
                    <img ... /></figure>
              <para>"As "<cite>^ article.66.cicero.wax</cite>...</para>
            </item>
          </list>
        </para>
      </section>
    </section>
    <inproc.44.brutus.s1 @ section>
      <info><title>"Acknowledgements"</title></info>
      <para>"We would like to thank the editors of "
        <cite>^ journal.adm.v10.n1</cite> ...</para>
    </section>
  </content>
</inproceedings>
</bib>
```

Figure 4.11: Exemplary Data Term, Part II: Non-XML and XML-style Term Syntax

```
<element>
  <identifier>journal.adm</identifier>
  <label>journal</label>
  <attributes total="true" />
  <children ordered="false" total="true">
    <element>
      <label>title</label>
      <attributes total="true" />
      <children ordered="true" total="true"
        >Applied Data Management</children>
    </element>
    <element>
      <label>editors</label>
      <attributes total="true" />
      <children ordered="true" total="true">
        <element>
          <label>editor-in-chief</label>
          <attributes total="true" />
          <children ordered="true" total="true"
            >Titus Pomponius Atticus</children>
        </element>
        <element>
          <label>editor</label>
          <attributes total="true" >
            <attribute><label>region</label><value>Africa</value>
            </attribute>
          </attributes>
          <children ordered="true" total="true"
            >Marcus Aemilius Aemilianus</children>
        </element>
        <element>
          <label>editor</label>
          <attributes total="true" >
            <attribute><label>region</label><value>Gaul</value>
            </attribute>
          </attributes>
          <children ordered="true" total="true"
            >Aulus Hirtius<!-- --></children>
        </element>
        <element>
          <label>editor</label>
          <attributes total="true" >
            <attribute><label>region</label><value>Cilicia</value>
            </attribute>
          </attributes>
          <children ordered="true" total="true"
            >Marcus Tullius Cicero<!-- --></children>
        </element>
      </children>
    </element>
    <element>
      <label>publisher</label>
      <attributes total="true" />
      <children ordered="true" total="true"
        >Titus Pomponius Atticus</children>
    </element>
    <element>
      <label>volumes</label>
      <attributes total="true" />
      <children ordered="true" total="true">
        <element>
          <identifier>journal.adm.v10</identifier>
          <label>volume</label>
          <attributes total="true" />
          <children ordered="true" total="true">
            <element>
              <identifier>journal.adm.v10.n1</identifier>
              <label>number</label>
              <attributes total="true">
                <attribute><label>type</label><value>special-issue</value>
                  </attribute>
              </attributes>
              <children ordered="false" total="true">
                <element>
                  <label>title</label>
                  <attributes total="true" />
                  <children ordered="true" total="true"
                    >Data Processing Challenges in the Age of Wax Tablets</children>
                </element>
                <element>
                  <label>editorial</label>
                  <attributes total="true" />
                  <children ordered="true" total="true"
                    ><reference>articles.66.cicero.wax</reference></children>
                </element>
                <element>
                  <label>year</label>
                  <attributes total="true" />
                  <children ordered="true" total="true"
                    >60</children>
                </element>
                <element>
                  <label>month</label>
                  <attributes total="true" />
                  <children ordered="true" total="true"
                    >july</children>
                </element>
              </children>
            </element>
            <element>
              <identifier>journal.adm.v10.n2</identifier>
              <label>number</label>
              <attributes total="true" />
              <children ordered="false" total="true">
                <element>
                  <label>year</label>
                  <attributes total="true" />
                  <children ordered="true" total="true"
                    >60</children>
                </element>
                <element>
                  <label>month</label>
                  <attributes total="true" />
                  <children ordered="true" total="true"
                    >november</children>
                </element>
              </children>
            </element> <!-- number -->
          </children>
        </element> <!-- volume -->
      </children>
    </element> <!-- volumes -->
  </children>
</element> <!-- journal -->
```

Figure 4.12: Exemplary Data Term, Excerpt: pure XML syntax

we take a look at how the document element of an XML document is interpreted as an Xcerpt data term. For the most part that is very straightforward, i.e., elements are mapped to structured terms, character data, comments, and processing instructions to their respective form of content terms. However, three issues demand a closer look:

- **Transparent Reference Resolution:** One of the strengths of Xcerpt is the transparent resolution of references. However, when reading XML documents one must consider

  1. How is the *identifier* of an element (represented by a structured data term) specified? Following, [8] and [32], Xcerpt aims at support for the two standard mechanisms for defining element ID's: attributes of type ID (declared in a DTD or similar schema) and xml:id attributes. Currently, the first one is not available, as Xcerpt does not yet provide access to type information from a schema, cf. Issue 8.3.

     All xml:id attributes in XML documents are translated to identifiers for the appropriate structured term, if the document does not contain xml:id errors (cf.[32], Section 2), in which case xml:id attributes are handled as normal attributes.

  2. How is a *reference* to a (defined) identifier specified? Again, Xcerpt aims to support the standard mechanism, i.e., attributes of type IDREF or IDREFS. However, there are other (internal) links, e.g., HTML-style href attributes. Furthermore, not all such links are to be transparently resolved, as discussed above.

     In all cases, the query author can specify a view that resolves the references and then formulate the rest of the query on top of this view. However, such a view requires a recursive descent through the document structure and is not trivial to program. Thus, Xcerpt introduces a processing instruction as a convenience that address the most common cases where transparent references are needed:

     The syntax of the processing instruction follows [12]: the target name is xcerpt-resolve-reference and the content of the processing instruction is a list of "pseudo-attributes" (again following syntax and notion from [12]). The following pseudo-attributes are supported:

     attribute specifies the name of the referencing attribute, i.e., the attribute that contains the actual reference(s).

     on specifies the (local) name of the element whose attributes contain the reference(s) to be resolved.

     ns specifies the namespace of the element. May be omitted in which case only references on elements in the empty namespace are resolved.

     type specifies the type of reference. Currently, the values IDREF, IDREFS, and fragment are supported, indicating that the referencing attribute contains a single ID reference, multiple ID references, or a single HTML-style fragment indicator (e.g., `#tiro`) respectively.

     replace specifies whether the element carrying the referencing attribute is merely a placeholder for the referenced element and thus is to be replaced by the reference. Possible values are true and false, with false as default value.

     E.g., the processing instruction

```
<?xcerpt-resolve-reference "attribute='idref' on='cite' type='IDREF'
                  ns='http://example.org/ns/docbook/simplified/1.0'"?>
```

specifies that all values in idref attributes on cite elements in the specified namespace are to be considered as IDREF links and transparently resolved when loading the document.

- **Unordered Content:** Though the children of structured terms are always ordered, Xcerpt allows the specification whether this order is significant and must be preserved. In XML documents this distinction can be made by annotating elements with the ordered attribute from the Xcerpt namespace (`http://xcerpt.org/ns/core/1.0`). The possible values are true or false, as in the pure XML syntax, indicating significant and insignificant order.

- **In-scope Namespaces:** XML documents provide no means to separate the scope of namespaces from the scope of individual elements. E.g., the content element in our sample data may contain elements from the XHTML namespace or from the namespace for our simplified Docbook version. However, the namespace declaration must be attached to individual elements, thus requiring either a wrapper element (the body element in line 77 in Figure 4.11 and line 89 in Figure 4.13) or separate namespace declarations on all sub-elements of content, cf. line 119 and 135 in Figure 4.13.

Figure 4.13 shows an XML document with the appropriate processing instructions and IDs to result in the sample data term, when loaded in Xcerpt.

```
<?xml version="1.0" standalone="yes"?>
<?xcerpt-resolve-reference
  "attribute='idref' on='cite' ns='http://example.org/ns/docbook/simplified/1.0'
   type='IDREF'"?>
<?xcerpt-resolve-reference "attribute='idref' on='in' type='IDREF"?>
<?xcerpt-resolve-reference "attribute='idref' on='editorial' type='IDREF"?>
<?xcerpt-resolve-reference "attribute='idref' on='ref' replace='true'
      type='IDREF"?>
<bib xmlns:xc="http://xcerpt.org/ns/core/1.0" xc:ordered="false">
  <journal xml:id="journal.adm" xc:ordered="false">
    <title>Applied Data Management</title>
    <editors>
      <editor-in-chief>Titus Pomponius Atticus</editor-in-chief>
      <editor region="Africa">Marcus Aemilius Aemilianus</editor>
      <editor region="Gaul">Aulus Hirtius
        <affiliation>Governor, Transalpine Gaul</affiliation>
      </editor>
      <editor region="Cilicia">Marcus Tullius Cicero
        <affiliation>Governor, Cilicia</affiliation></editor>
    </editors>
    <publisher>Titus Pomponius Atticus</publisher>
    <volumes>
      <volume xml:id="journal.adm.v10">
        <number xml:id="journal.adm.v10.n1" type="special-issue"
              xc:ordered="false">
          <title>Data Processing Challenges in the Age of Wax
            Tablets</title>
          <editorial idref="articles.66.cicero.wax"></editorial>
          <year>60</year>
          <month>july</month>
        </number>
        <number xml:id="journal.adm.v10.n2" xc:ordered="false">
          <year>60</year>
          <month>november</month>
        </number>
      </volume>
    </volumes>
  </journal>
  <proceedings xml:id="conf.dmmc" xc:ordered="false">
    <editors>
      <editor>Marcus Aemilius Lepidus
        <affiliation>Consul, SPQR</affiliation></editor>
      <editor>Gaius Julius Caesar Octavianus</editor>
      <editor>Marcus Antonius</editor>
    </editors>
    <title>Advancements in Data Management for Military and Civil
      Application</title>
    <invited-papers>
      <ref idref="inproc.44.brutus" />
      <ref idref="article.66.scaurus.qumran" />
    </invited-papers>
    <abbrev>DMMC</abbrev>
    <year>44</year>
    <month>july</month>
    <location>Mutina</location>
    <publisher>SPQR</publisher>
  </proceedings>
  <article xml:id="article.66.scaurus.qumran" xc:ordered="false">
    <author>Marcus Aemilius Scaurus
      <affiliation>Tribun, Gnaeus Pompeius Magnus</affiliation>
    </author>
    <title>From Wax Tablets to Papyri: The Qumran Case Study</title>
    <in scrolls="102-112" idref="journal.adm.v10.n1" />
    <citations>
      <cite ref="article.66.cicero.wax" />
      <cite type="formatted">M. Aemilius Scaurus (104): A Case for
         Permanent Storage of Senate Proceedings. In: M. Aemilius
         Scaurus, ed. (104): <i>Princeps Senatus: Honor
         and Responsibility</i>, Chapter 2, 14-88.</cite>
      </citations>
    </article>
    <article xml:id="article.66.cicero.wax" xc:ordered="true">
      <authors>
        <author>Marcus Tullius Cicero
          <affiliation>Governor, Cicilia</affiliation></author>
        <author>Marcus Aemilius Lepidus
          <affiliation>Gens Aemilia</affiliation></author>
        <author>Marcus Tullius Tiro
          <affiliation>Secretary, M. T. Cicero</affiliation>
        </authors>
        <title>Space- and Time-Optimal Data Storage on Wax Tablets</title>
        <in scrolls="1-94" idref="journal.adm" />
        <content type="xhtml">
          <body xmlns="http://www.w3.org/1999/xhtml">
            <!-- incomplete due to melted letters on some tablets -->
            <h1 id="contributions">Contributions</h2>
            <h1>A History of Data Storage: From Stone to Parchment</h1>
            <p>Despite recent evidence ...</p>
            <ol>
              <li><em><strong>Homeric</strong> Age:</em>...</li>
              <li><em>Age of the <strong>Kings</strong>:</em>...</li>
            </ol>
            <h1 id="tiro">Notae Tironianae</h1>
            <img title="Tironian et" src=... />
            <p>As discussed in <a href="#contributions">...</a></p>
            <h1 id="tachygraphy">Challenges for Tachygraphy on Wax</h1>
            <p>Though conditions for writing on wax tablets are adverse to
            tachygraphy, systems as described in <a href="#tiro">...</a></p>
          </body>
        </content>
      </article>
      <inproceedings xml:id="inproc.44.brutus" xc:ordered="false">
        <authors>
          <author>Marcus Antonius<affiliation>Consul, SPQR</affiliation>
          </author>
          <author>Decimus Junius Brutus<affiliation
               >Governor, Cisalpine Gaul</affiliation></author>
        </authors>
        <title>Efficient Management of Rapidly Changing Personal
             Records</title>
        <in scrolls="24-48" idref="conf.dmmc"/>
        <content type="docbook">
          <section xmlns="http://example.org/ns/docbook/simplified/1.0">
            <info><title>Introduction</title></info>
            <section><info><title>Contributions</title></info>
              <para>The most notable contributions of this article include:
                <list type="ordered">
                  <item>
                    <para>A new <em>methodology</em> to ..., cf.
                      <pageref idref="inproc.44.brutus.s1" /> ...</para>
                    <figure><title>Chart of Desertions</title>
                          <img ... /></figure>
                    <para>As <cite idref="article.66.cicero.wax" />...</para>
                  </item>
                </list>
              </para>
            </section>
          </section>
          <section xml:id ="inproc.44.brutus.s1"
                 xmlns="http://example.org/ns/docbook/simplified/1.0">
            <info><title>Acknowledgements</title></info>
            <para>We would like to thank the editors of
              <cite idref="journal.adm.v10.n1" /> ...</para>
          </section>
        </content>
      </inproceedings>
    </bib>
```

Figure 4.13: Exemplary Data Term: From an XML Document

Chapter 5

How to specify queries? Part 1: Construction

As briefly mentioned above, Xcerpt uses very much similar concepts and syntax for data and queries. Queries in Xcerpt are guided by a small number of principles:

- **Queries as Patterns.** Instead of using separate concepts and syntax for queries (as in navigational query languages such as XQuery [6]), Xcerpt uses terms for representing both data and queries. All data terms are also query terms, but there are some additional constructs in data terms, that allow (a) the extraction of data by using logical variables, (b) the specification of queries that are only incomplete patterns of the data, i.e., where more nodes may occur in the data than specified in the query, and (c) the specification of formulas in terms, i.e., conjunction, disjunction, negation, optionality etc.

- **Logical Variables.** In query terms, logical variables are used to indicate which data is to be selected and to join data (indicated by multiple occurrences of the same variable as in logic programming languages). The result of a query is conceptually a set of tuples each representing a combination of bindings (or matches) for all the variables occurring in the query term. For each tuple, a data term must exist that matches the query where all the variables are substituted by the bindings of the tuple.

- **Separation of Querying and Construction.** In contrast to query languages such as SQL or XQuery, construction and querying are strictly separated in Xcerpt, in particular there are no nested queries in Xcerpt (rather rules and rule chaining is used, cf. Section 7). The data constructed by a rule is specified in construct terms, that contain variables from the corresponding query terms acting as placeholders for selected data. Additionally construct terms make use of gouping constructs to return all or some of the alternative bindings of a variable.

- **Incomplete Patterns.** In most cases, queries specify just enough restrictions on the data to be returned, as required by the query intent, rather than specifying full or “total” patterns of the data. Xcerpt supports such queries by providing constructs to express that a pattern is incomplete in breadth (i.e., there can be more children than specified), depth (i.e., there can be additional nodes and edges between the matched nodes) etc.

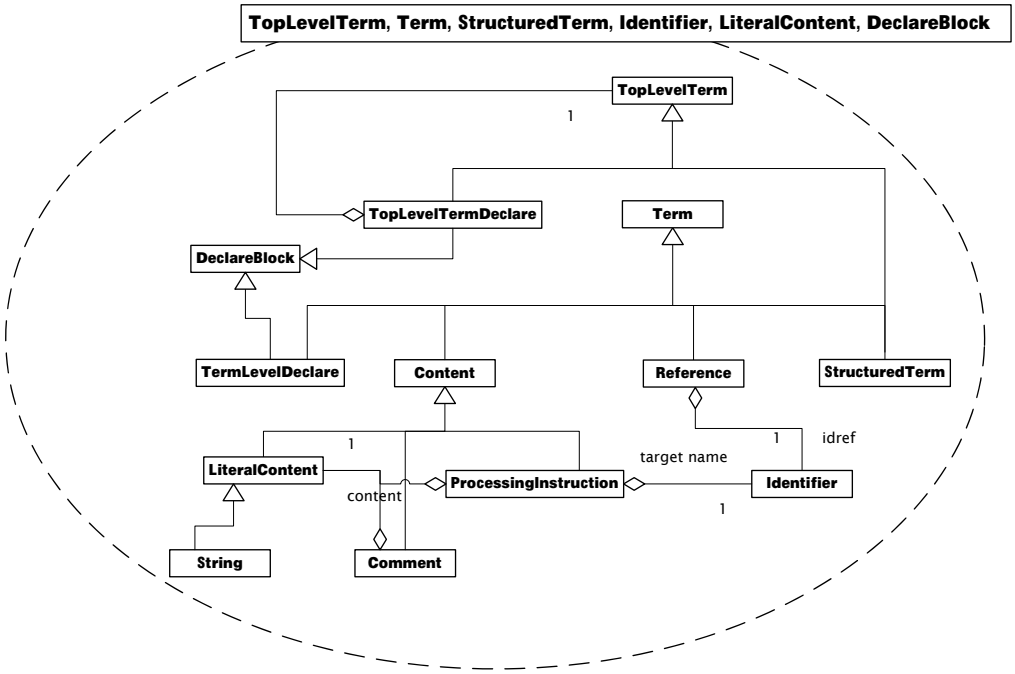


Figure 5.1: UML Model for Terms as Parameterized Collaboration

- **Terms as Formulas.** Query terms are not only augmented by variables, but also by constructs for expressing negation, disjunction, conjunction, and optionality.

In the remainder of this part, first construct terms and then query terms are introduced in detail and compared to data terms. To facilitate a better understanding and description of the differences between data terms, construct terms, and query terms, a short aside introduces a parameterized model for terms, that precisely specifies in what aspects the three kinds of terms may differ from each other.

5.1 An Aside: A Parameterized Model for Terms

UML uses the notion of **"parameterized collaborations"** to describe what is otherwise known as (software) *patterns* (not to be confused with Xcerpt's patterns), i.e., collections of concepts and relations among concepts that occur in different contexts. They are "parameterized", as a number of the concepts in the parameterized collaboration are "exported" as parameters and must be related to concrete concepts when using the pattern.

Figure 5.1 shows an example for the notation adopted in UML for defining such parameterized collaborations: concepts and relations are drawn as usual, but a *dashed ellipsis* is drawn around the concepts that are part of the definition. The parameter concepts are depicted in a *box* at the top of the ellipsis.

Indeed, Figure 5.1 shows an Xcerpt term as a parameterized collaboration: all the relations

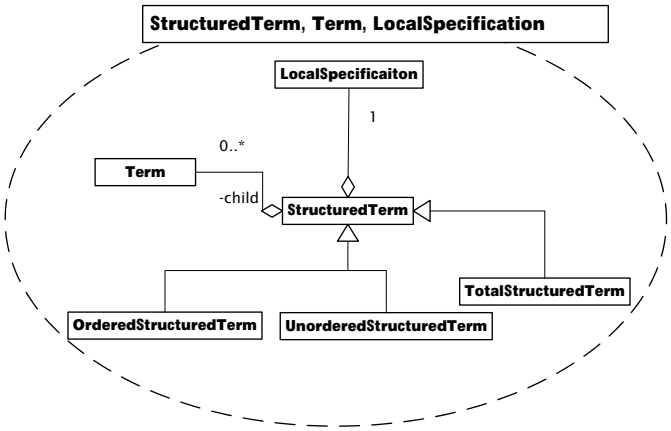


Figure 5.2: UML Model for Structured Terms as Parameterized Collaboration

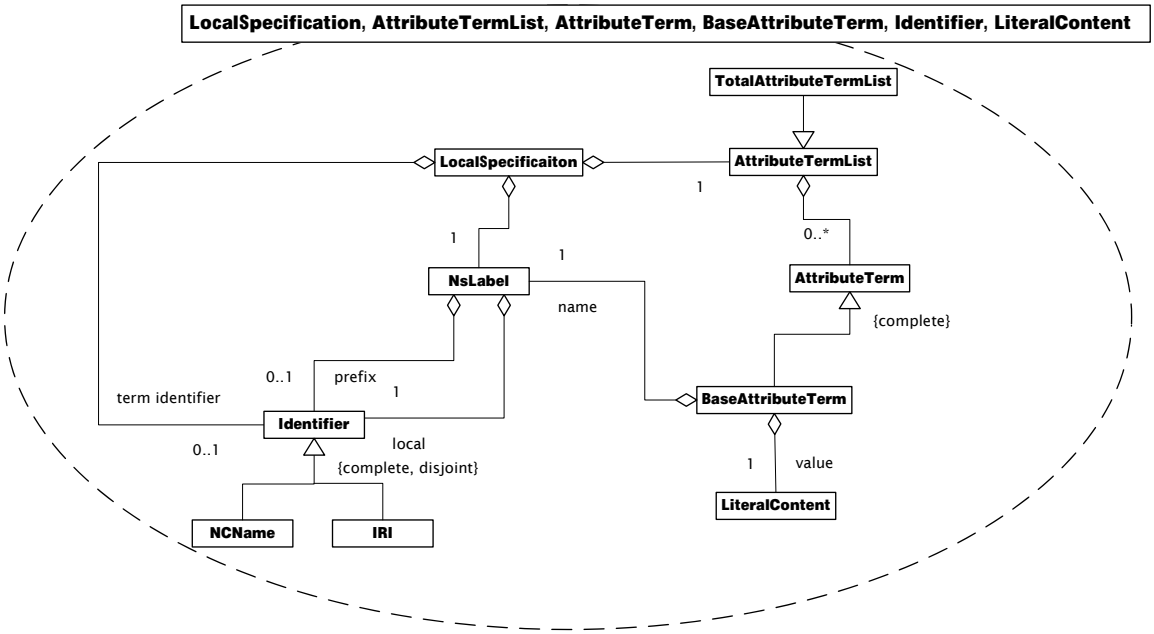


Figure 5.3: UML Model for Local Term Specifications

and concepts depicted are common to all three kinds of Xcerpt terms, they only vary in the six parameters given in the top corner: (1) what is a top-level term, (2) what is a basic term, (3) what is a structured term, (4) what is an identifier, (5) what is a literal content, and (5) what is a declaration block.

Figures 5.2 and 5.3 complete the definition of a term by defining parameterized collaboration for structured terms and for local descriptions of terms (these could be part of a single pattern as they are never used separately, but for readability they have been split over three diagrams). The parameterized collaboration for structured terms shares the second and third parameter of the parameterized collaboration for terms and adds an additional parameter, the local specification, to link to the parameterized collaboration for local term specifications shown in Figure 5.3. The parameterized collaboration for a local term specification has additional parameters for (1) attribute terms, (2) attributes, and (3) basic (or literal) attributes.

Notice, how similar these patterns are to data terms. This is due to the fact, that all data terms are also valid terms in the other two term kinds.

Given these parameterized collaborations, data terms can be defined as shown in Figure 5.4: all parameters for the three parameterized collaborations are simply "instantiated" with concrete concepts for data terms without adding *any* additional concepts or relations.

5.2 Specifying New Data: Construct Terms

As mentioned above, conceptually the result of a query is a *multi-set of mappings* each representing one combination (or *substitution*) of bindings for all variables occurring in the query term. For each tuple, an (extensional or intensional) data term must exist that matches the query where all the variables are substituted by the bindings of the tuple.

5.2.1 Substitutions and Substitution Sets

A *substitution* is a mapping from the set of (all) variables to the set of (all) construct terms. As usual, a substitution is a mapping of infinite sets. Of course, finite representations are usually used, as the number of variables occurring in a term is finite. Substitutions are often conveniently denoted as sets of variable assignments instead of as functions. For example, we write $\{\mathsf{X} \mapsto a, \mathsf{Y} \mapsto b\}$ to denote a substitution that maps the variable X to a and the variable Y to b, and any other variable to arbitrary values. In general, a substitution provides assignments for all variables, but "irrelevant" variables are not given in the description of substitutions.

A *substitution multi-set* is simply a multi-set containing substitutions. Often the substitutions in a substitution multi-set have very similar sets of "relevant" variables, differing only, e.g., in optional variables. Thus a substitution multi-set can also be denoted as an n-ary multi-set relation over the set of all construct terms where n is the size of the maximum set of variables "relevant" for any substitution in the multi-set. Substitutions become tuples in this relation with "irrelevant" variables marked as `null` values. E.g., the following table is a representation for a substitution multi-set with three substitutions using "relevant" variables X, Y, and Z. The first tuple represents the substitution $\{\mathsf{X} \mapsto a, \mathsf{Y} \mapsto b, \mathsf{Z} \mapsto c\}$, the second $\{\mathsf{X} \mapsto c, \mathsf{Y} \mapsto b, \mathsf{Z} \mapsto b\}$, and the third $\{\mathsf{X} \mapsto c, \mathsf{Y} \mapsto a\}$.

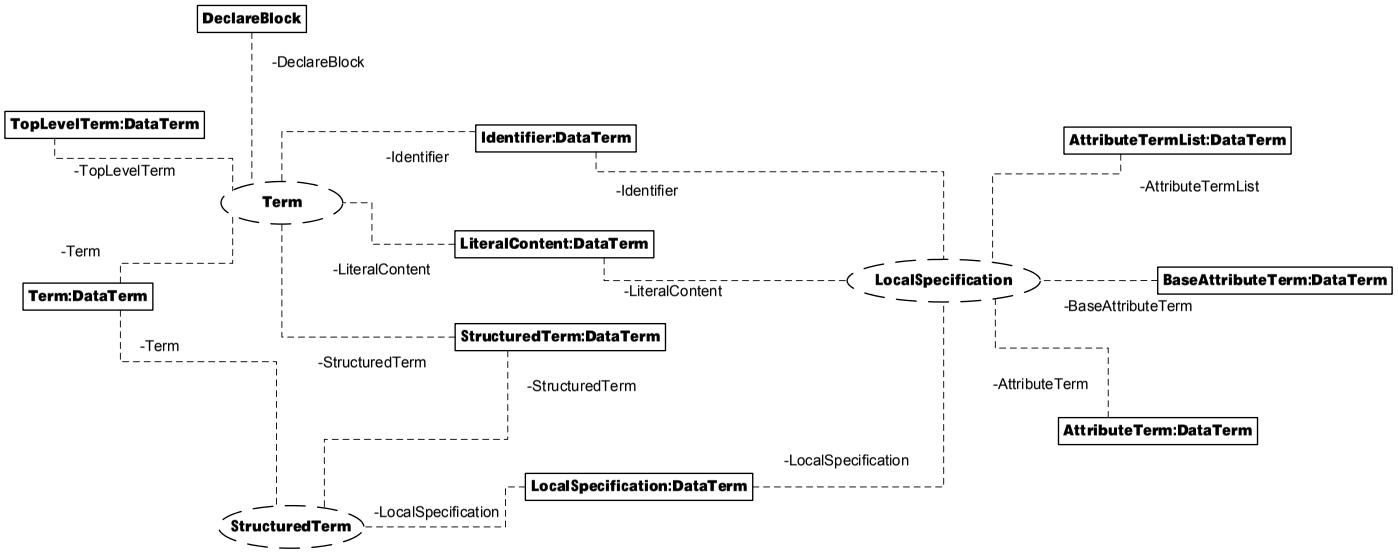


Figure 5.4: UML Model for Data Terms using Parameterized Collaborations, cf. Figure 5.1–5.3

X	Y	Z
a	b	c
c	b	b
c	a	**null**

More details on substitutions can be found in [37, 38]. Notice, that there substitution sets are used to simplify definitions and proofs (cf. Issue 1).

5.3 The Shape of Construct Term

§1 Construct term

Construct terms specify *the shape of data that is constructed (or derived)* for each match of the corresponding query term. In that, they are comparable to clause heads in Datalog.

If a construct term contains no variables, it becomes equivalent to a data term: The full shape of the derived data is specified. But construct terms may also contain *variables*, viz. in place of sub-terms (term variables) and in place of literal content or identifiers such as element labels and namespaces, attribute labels, etc. (literal variables).

However, there is one major difference between Xcerpt (and similar query languages for structured data) when comparing to Datalog on flat tuples: E.g., given bindings for authors and titles one would like to create author elements that contain for each author all corresponding titles. Explicit support for *grouping constructs* in the scope of other data is needed to express that form of construction.

Figure 5.5 shows an UML model for construct terms using the parameterized collaborations for general terms introduced in the aside of Section 5.1. The figure highlights the exact differences between data and construct terms:

1. **Variables** can occur instead of (a) (structured or attribute) terms or instead of (b) identifiers and literal content.

2. **Modifiers** specify (a) the grouping of sub-terms by one or more variables, i.e., the repetition of parts of a construct term for all or some of the alternative bindings of one or more variables, and (b) the optionality of sub-terms, i.e., the omission of a part of a construct term based on the bindings of one or more variables.

Notice, that the functionality of these modifiers is almost a corollary of adding variables: Once variables that may have more than one binding are allowed in construct terms, it is necessary to handle the case of bindings for one variable included in construct terms for bindings of another one (grouping). In the same way, once variables may have no bindings at all, it is necessary to define which part of a construct term is to be left out if there is no bindings (optionality).

Figures 5.6 and 5.7 detail modifiers for structured and attribute terms: All modifiers "modify" construct terms to indicate that the modified term is to be handled differently from its unmodified form. The construct terms modified by a modifier are the *scope of the modifier*. In construct terms, all modifiers have a scope of *one or more* construct terms except the grouping modifier for attribute terms. The latter one has a scope of a single attribute construct terms and deviates from the general rule, as attributes are unordered and single-valued (i.e., there may be no repeated attribute names) and thus grouping over sequences of terms is not useful.

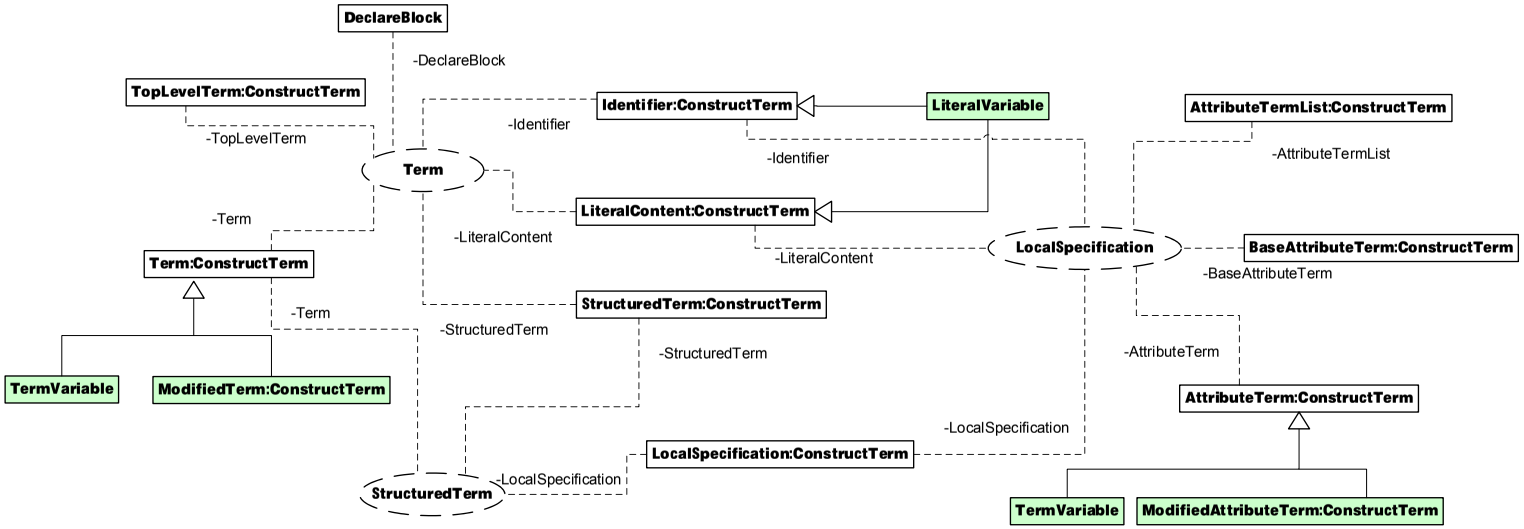


Figure 5.5: UML Model for Construct Terms using Parameterized Collaborations

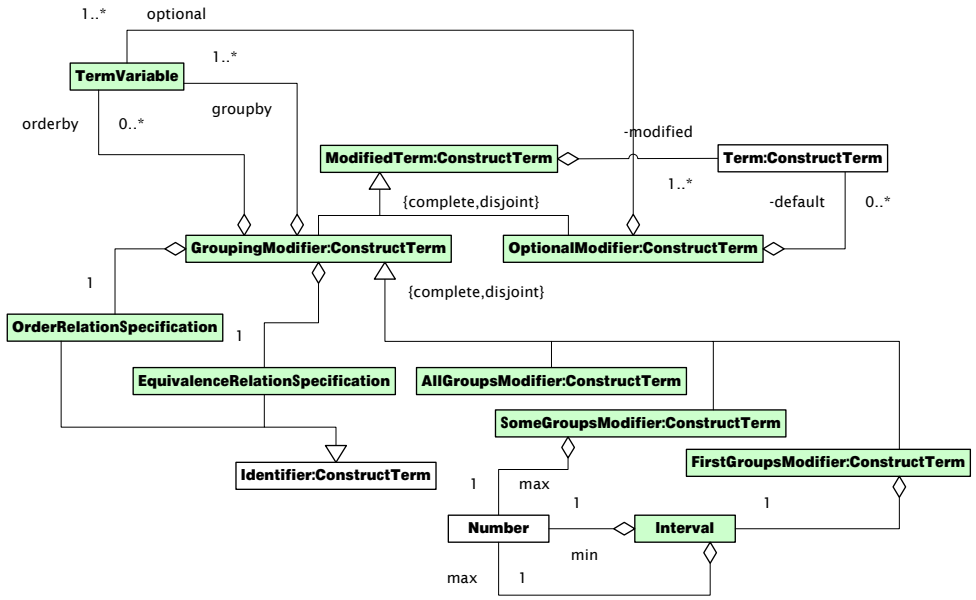


Figure 5.6: UML Model for Modified Structured Construct Terms

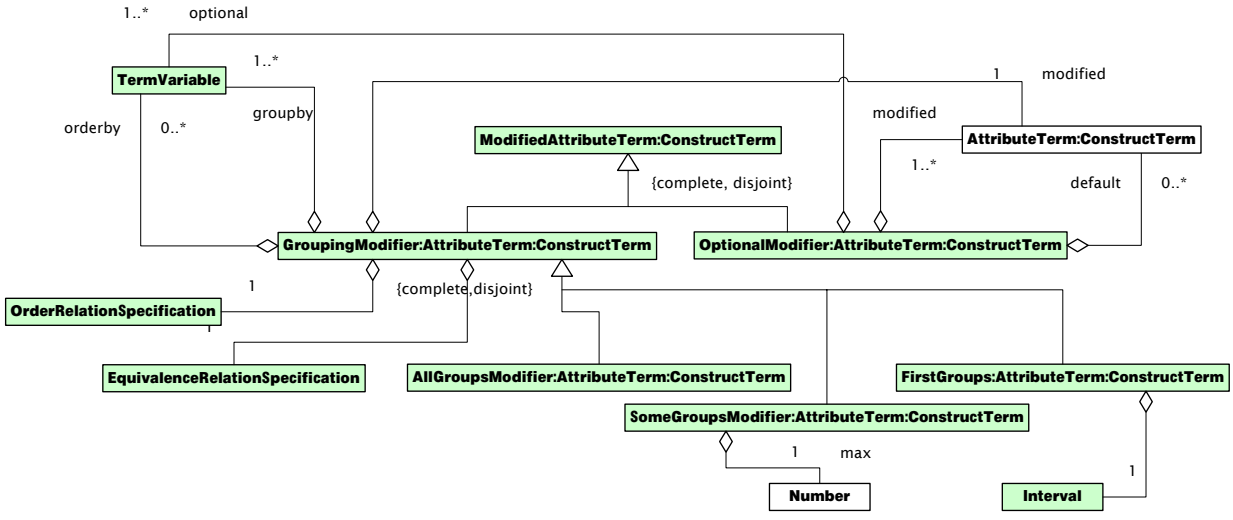


Figure 5.7: UML Model for Modified Attribute Construct Terms

5.3.1 Textual Term Syntax

Although the syntactic differences between data and construct terms are from a conceptual perspective few, the EBNF specification of the non-XML (as well as of the XML-style) term syntax share only the productions for lexical structures. This is due to the inability of the EBNF notation to express parameterized productions or grammars. E.g., declaration blocks are identical except that in data terms they contain data terms and in construct terms they contain construct terms. In EBNF two separate non-terminals (and productions) are needed, since it is not possible to parameterize the production of declaration blocks with respect to the kind of contained terms.

Aside of this minor nuisance, the productions are very similar. In fact, except for the following four productions and the addition of variables and modifiers discussed below they are identical and will not be repeated in detail here. With the exception of the following four non-terminals and their productions all productions are copied from the data term syntax replacing the `-dt` prefix in all non-terminals by `-ct`. The full syntax can be found in Appendix A.3.

The deviation lies in the productions for the non-terminals ⟨*construct-term*⟩ and ⟨*attr-term-ct*⟩, where term variables and modified terms (for grouping and optionality) are added, as well as for ⟨*identifier-ct*⟩ and ⟨*literal-content-ct*⟩, where literal variables are added:

⟨*construct-term*⟩ ::= ⟨*term-level-declare-ct*⟩ | ⟨*reference-ct*⟩ | ⟨*structured-ct*⟩ | ⟨*content-ct*⟩ | ⟨*term-variable-ct*⟩ | ⟨*modified-ct*⟩

⟨*attr-term-ct*⟩ ::= ⟨*base-attr-term-ct*⟩ | ⟨*term-variable-ct*⟩ | ⟨*modified-attr-term-ct*⟩

⟨*identifier-ct*⟩ ::= ⟨*NCName*⟩ | ⟨*IRI*⟩ | ⟨*String*⟩ | ⟨*literal-variable-ct*⟩

⟨*literal-content-ct*⟩ ::= ⟨*String*⟩ | ⟨*literal-variable-ct*⟩ ⟨*String*⟩ | ⟨*literal-variable-ct*⟩

Though conceptually, literal and term variables are different and not always interchangeable they are not separated syntactically, as this is left to as-of-now unfinished type system. Syntactically variables are represented in the term syntax by names preceded with the keyword **`variable`** (or its shorthand `var`). A variable may occur without the leading keyword, if it is in the scope of a variable declaration that reserves the name of the variable, cf. Section 6 and 7.

⟨*variable-ct*⟩ ::= ⟨*term-variable-ct*⟩ | ⟨*literal-variable-ct*⟩

⟨*term-variable-ct*⟩ ::= ⟨*var-specification-ct*⟩

⟨*literal-variable-ct*⟩ ::= ⟨*var-specification-ct*⟩

⟨*var-specification-ct*⟩ ::= `'variable'` ⟨*NCName*⟩ | `'var'`

Finally, modifiers occurring in construct terms can be distinguished in grouping and optional modifiers, the detailed syntax of which is discussed in the following sections.

⟨*modified-ct*⟩ ::= ⟜ ⟨*grouping-ct*⟩ - ⟨*optional-ct*⟩ ⟶

⟨*modified-attr-term-ct*⟩ ::= ⟜ ⟨*grouping-attr-term-ct*⟩ - ⟨*optional-attr-term-ct*⟩ ⟶

Notice, that the grammar does not specify any order among the modifiers, i.e., both a grouping modifier in the scope of an optional modifier and vice versa are allowed.

5.3.2 XML-style Term Syntax

The XML-style term syntax once again uses the same productions as the non-XML term syntax, deviating from data terms (in XML-style term syntax) in exactly the same ways. The full grammar is shown in Appendix B.3.

5.3.3 Pure XML Syntax

In contrast to EBNF notation, Relax NG provides some means for parameterized grammars, as used above for defining declaration blocks. Indeed, construct terms can be defined using the same grammar as for data terms, but parameterizing term.class and attribute.term.class, as well as identifier.class and literal-content.class. In all cases the parameterization happens by adding additional choices to the existing ones: variables and modified terms or attributes.

Then definitions for variables as well as modified terms and attributes needs to be added to the grammar. Modified terms and attributes are defined using another parameterized grammar, this time the parameters are the content of the modifiers (once structured construct terms, once attribute construct terms) and what represents a variable. The details of that grammar are discussed in the following section.

```
construct-term =
 grammar {
  variable-ct = parent variable-ct
  # Add grouping and optional for attributes
  modified-attribute =
   grammar {
    include "modifiers.rnc" {
     start = grouping
     content = parent attribute-term.class
     variable = parent variable-ct
    }
   }
   | grammar {
     include "modifiers.rnc" {
      start = optional
      content = parent attribute-term.class*
      variable = parent variable-ct
     }
    }
  # Add grouping and optional for elements
  modified-term =
   grammar {
    include "modifiers.rnc" {
     start = grouping
     content = parent term.class*
     variable = parent variable-ct
    }
```

```
    }
29  | grammar {
      include "modifiers.rnc" {
31      start = optional
        content = parent term.class*
33      variable = parent variable-ct
      }
35  }

37  ## Construct terms may also be variables or modified by
    ## grouping and optional modifiers.
39  term.class |= variable-ct | modified-term

41  ## Construct attribute terms may also be variables or modified by
    ## grouping and optional modifiers.
43  attribute-term.class |= variable-ct | modified-attribute
    # Add variables to identifiers and literal content
45  identifier.class |= variable-ct
    literal-content.class |= variable-ct
47  include "term.rnc"
}
```

5.4 Grouping in Construct Terms

§2 Grouping modifier

A grouping modifier expresses a grouping over the bindings of all its *grouping variables*. For alternative substitutions of the grouping variables, the construct in the scope of the grouping modifier are *repeated* once with the occurrences of the variables substituted accordingly.

A grouping modifier specifies four aspects of a grouping:

1. **Scope: What is to be repeated?** The scope of grouping modifiers for structured terms is a list of construct terms that is to be repeated. This allows, e.g., the bracketing of grouped terms or the creation of structures such as sections in HTML that are expressed through element delimiters instead of nesting. Grouping modifiers for attributes, however, only apply to a single attribute term, e.g., a variable or an attribute specification containing a variable for the name of the attribute. Lists of terms are not useful in this case, as attributes are always unordered and no two attributes of the same element node may have the same name.

2. **Groups: How to form groups?** An essential part of grouping is the determination of the actual groups: i.e., to specify when two substitutions of the associated grouping variables are considered **equivalent** and thus part of the same "group" (i.e., equivalence class).

   Commonly, query languages use a *single, pre-defined equivalence relation* for grouping, e.g., SQL uses equivalence based on the *typed value* of the grouping attributes, i.e., all tuples with the same typed value for the grouping attributes are considered as one group. In object-oriented or semi-structured query languages, one as finds equivalence based on object or *node identity*, i.e., substitutions for the grouping variables are considered equivalent only if they have for each grouping variable the very same nodes as substitution.

In Xcerpt, the default equivalence relation is *structural equivalence*, i.e., two bindings are considered equivalent if their label, children, and/or content is equal (formally, structural equivalence in Xcerpt uses the notion of simulation as defined in [37, 11], cf. Section 6.1). Beyond this default equivalence, Xcerpt's grouping modifiers may also explicit specify an **equivalence relation** that relates equivalent substitutions for the grouping variables. It must adhere to the usual definition of an equivalence relation, i.e., it must be a reflexive, symmetric, and transitive relation over the domain

3. **How to order the repeated terms?** An order among the groups established in point (2) is needed not only in the case of **`first`**-selecting grouping terms, but also if the grouping term is contained in a (structured) term where the order of the children is significant. In both cases, the order among the groups is defined by (a) a list of *ordering variables* and (b) a *total order relation* (i.e., a reflexive, antisymmetric, transitive, and comparable relation) on the substitutions for ordering variables. Note, that the ordering variables must be a subset of the grouping variables. Also note, that the order relation must be consistent with the equivalence relation, i.e., whenever $b_1 \leq b_2$ and $b_2 \leq b_1$ for the order relation $\leq$ use in a grouping term G and bindings b_1, b_2, then $b_1 \sim b_2$ for the equivalence relation $\sim$ used in G.

4. **Group Selection: Which of the groups to consider?** In many cases, the grouping should only iterate over certain of the groups. Xcerpt addresses the selection of relevant groups by providing three grouping modifiers:

   - The **`All`**-Groups modifier uses *all* of the groups established as explained in point (2).
   - The **`Some`**-Groups modifier uses *some* of the groups: At most m groups are selected *arbitrarily* and possibly non-deterministically. *At most* m, as there may be less than m groups, in which case, all groups are selected.
   - The **`First`**-Groups modifier uses *some* of the groups, but the selection is determined by the order of the groups: An interval $n - m$ specifies that the n^{th} to m^{th} group are to be used. Again, there may be less than m (in which case all groups after and including the n^{th} group are used) and even less than n groups (in which case no group is used). The order of the groups is defined by the order relation described in the previous point.

Grouping modifiers may be *nested* leading to the expected behavior: say a grouping over authors of books contains another grouping over titles of books. In the constructed data, the terms constructed by the grouping over titles are contained in the terms constructed by grouping over authors based on the author-title combinations found in the substitutions. Intuitively, nested grouping constructs are similar to nested **`for`**-loops in imperative programming languages.

To summarize Xcerpt's grouping modifiers allow the repetition of subterms based on substitutions for grouping variables. They allow extensive customization of what defines a group and how to order the repetitions without sacrificing simplicity in common cases.

Like in all query languages, where the result of a query can have a complex (structured) shape, *grouping is not only essential in combination with aggregation* (as in relational query languages), but also to define how the nesting of the result is constructed based on the relations of data items selected by a query in variable bindings. The nature of data with complex shape also requires the support of nested grouping, i.e., repetition within repetition.

5.4.1 Textual Term Syntax

The non-XML term syntax for grouping modifiers in construct terms closely reflects the four aspects of the abstract syntax:

1. *Group selection* is indicated using the three different keywords **`all`**, **`some`**, **`first`**. **`some`** is followed by a number (or a variable) that indicates the number of groups to select. **`first`** is followed by an interval specification, i.e., two numbers (or variables) separated by a $-$. Two shorthands for intervals are provided: $n-$ to select all groups starting with the n^{th} and $+$ as abbreviation for $1-$. Thus **`first`** $1-$ is equivalent to **`all`**.
2. The *scope* of the modifiers are the construct terms included in parentheses after the modifier. As in declaration blocks the parentheses may be omitted, if the scope is exactly one construct term.
3. *Groups* are formed using the optional equivalence relation on the bindings of the grouping variables. *Grouping variables* are either implicit or explicit. Implicit grouping variables are all free variables in the scope of the grouping modifier, i.e., all variables that occur in the scope of the grouping modifier but not in the scope of another nested grouping modifier. Explicit grouping variables are specified in a list enclosed by parentheses after the **`order-by`** keyword. Again the parentheses may be omitted if the list is a singleton.
4. The *order* of the groups is determined by the order variables (a subset of the grouping variables) and the order relation. The order variables are specified in a list (enclosed by parentheses) after the keyword **`order-by`**. Again the parentheses may be omitted if the list is a singleton. Notice, that if both are present **`order-by`** follows **`group-by`**.

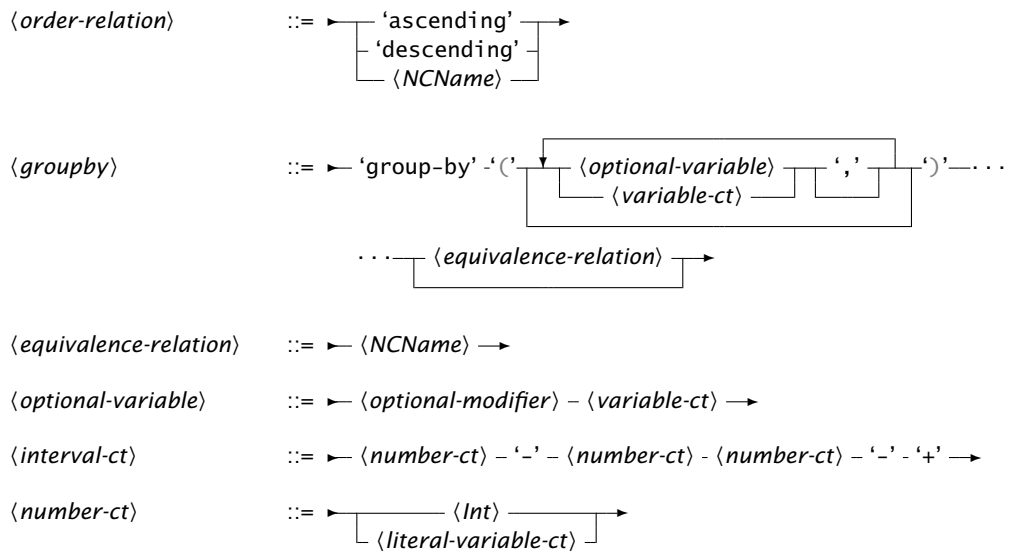
For attributes the specification is similar, but only a single attribute construct term may in the scope of a grouping modifier. This prevents the repetition of same-name attributes (recall, that attributes are essentially (key, value) pairs in a dictionary associated with their structured term and duplicate keys are forbidden in accordance to XML).

⟨*grouping-ct*⟩ ::= ⟨*grouping-modifier*⟩ '(' ⟨*construct-term*⟩ ',' ')' ... ⟨*groupby*⟩ ⟨*orderby*⟩

⟨*grouping-attr-term-ct*⟩ ::= ⟨*grouping-modifier*⟩ '(' ⟨*attr-term-ct*⟩ ')' ... ⟨*groupby*⟩ ⟨*orderby*⟩

⟨*grouping-modifier*⟩ ::= 'all' | 'some' ⟨*number-ct*⟩ | 'first' ⟨*interval-ct*⟩

⟨*orderby*⟩ ::= 'order-by' '(' ⟨*optional-variable*⟩ | ⟨*variable-ct*⟩ ',' ')' ... ⟨*order-relation*⟩



5.4.2 XML-style Term Syntax

The same productions as for the non-XML term syntax can be used for the XML-style term syntax. The full grammar is given in Appendix B.3.

5.4.3 Pure XML Syntax

As seen above, the pure XML syntax can utilize parameterized grammars not just for construct terms in general, but also for modifiers itself. Figure 5.8 shows the Relax NG schema for that grammar. As in the grammar for declaration blocks, the content pattern is to be overwritten when importing this grammar. Additionally also the variable pattern can be replaced to specify the shape of variable occurrences.

The following listing gives the textual grammar in Relax NG's compact syntax:

```
default namespace = "http://xcerpt.org/ns/core/1.0"
namespace a = "http://relaxng.org/ns/compatibility/annotations/1.0"

start = grouping
content = empty
grouping =
  element all { content, order-by?, group-by? }
  | element some { number, content, order-by?, group-by? }
  | element first { interval, content, order-by?, group-by? }
order-by =
  element order-by {
    attribute order-relation { text }?,
    optional-variable+
  }
group-by =
  element group-by {
    attribute equivalence-relation { text }?,
    optional-variable+
  }
optional-variable =
```

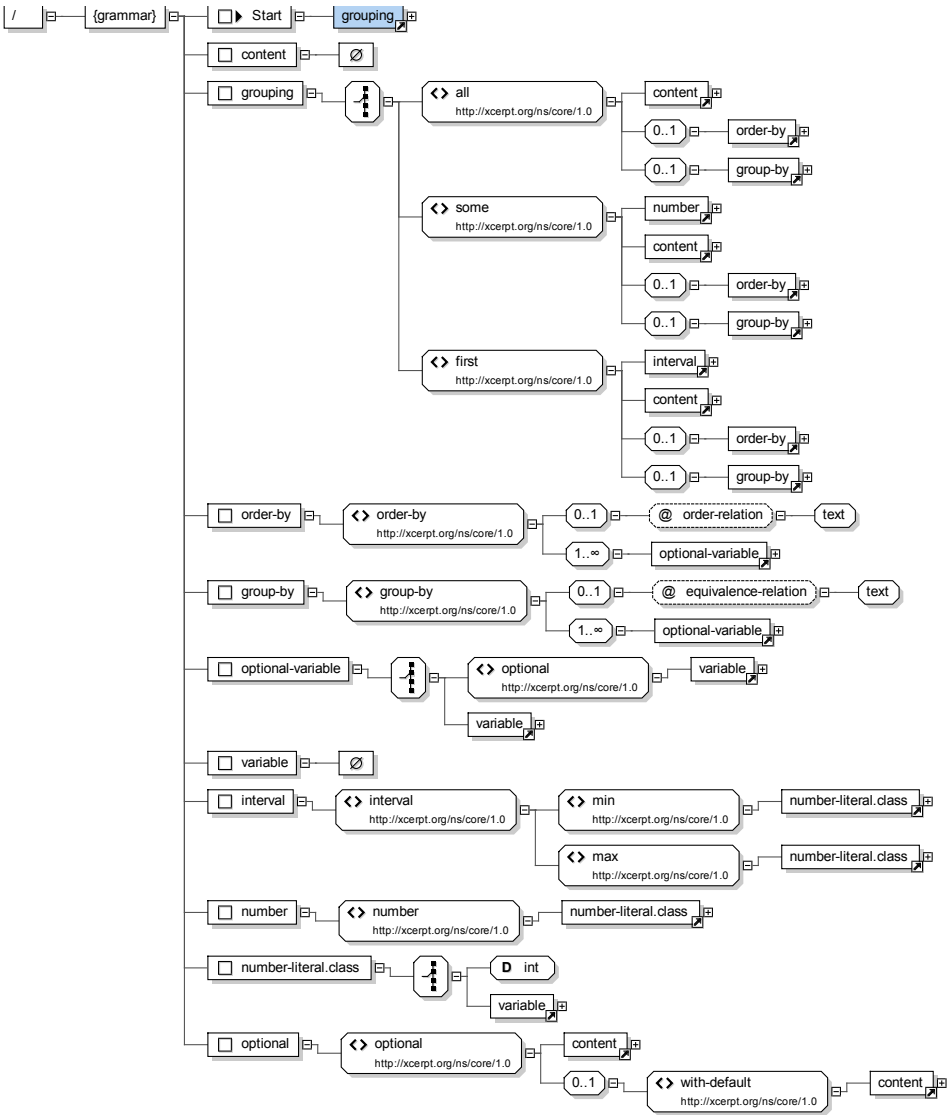


Figure 5.8: Relax NG Grammar for Modifiers in Construct Terms

```
    element optional { variable }
22   | variable
   variable = empty
24 interval =
    element interval {
26    element min { number-literal.class },
      element max { number-literal.class }
28   }
   number = element number { number-literal.class }
30 number-literal.class = xsd:int | variable
   optional =
32   element optional {
       content,
34     element with-default { content }?
     }
```

As in the term syntax, grouping variables can be specified explicitly or implicitly, i.e., in the group-by subelement or by occurring as free variables inside the scope of the grouping modifier.

5.5 Optional Construct Terms

Aside of grouping to collect alternative bindings for a variable, Xcerpt's construct terms add one more modifier to handle the case where a variable may have no bindings in a given substitution: the optional construct term.

§3 Optional Modifier

An optional modifier in construct terms specifies a form of *conditional construction*: some of the variables reported as result of a query may not have any bindings in some substitutions, as they occur only in optional parts of a query (cf. Section 6.1). In this case, an optional modifier must be used in the construct term to mark the part of the construct term that depends on the existence of bindings for the optional variables associated with the optional modifier.

Recall, Figures 5.6 and 5.7 for the precise model of optional construct terms. An optional modifier in construct terms needs three parameters, the first two are similar to those for grouping modifiers:

1. **Scope: Which part of the construct term is optional?** The scope of an optional modifier are the modified construct terms. Again, a list of construct terms is allowed to facilitate optional parts that cover several siblings, e.g., for bracketing. In contrast to grouping modifiers for attribute terms, optional modifiers have also in the case of attribute terms a *list* of (attribute) construct terms as scope, since the problem with repeated attribute names does not occur in this case (since there is no repetition).

   The construct terms in the scope of an *optional modifier are the only place, where optional variables may occur* in construct terms. More precisely, only those optional variables that an optional modifier O (or an optional modifier that O is part of) depends on, may occur in O.

2. **Condition: On bindings for which variables depends the conditional construction?** Optional modifiers specify a set of optional variables that are used to determine whether the construct terms in the scope of the optional modifier are part of the result or not:

They are included in the result for a substitution σ only if bindings for all the specified optional variables exist in σ. Note, that all variables that an optional modifier depends on *must be optional* in the query as well (cf. Section 6.1).

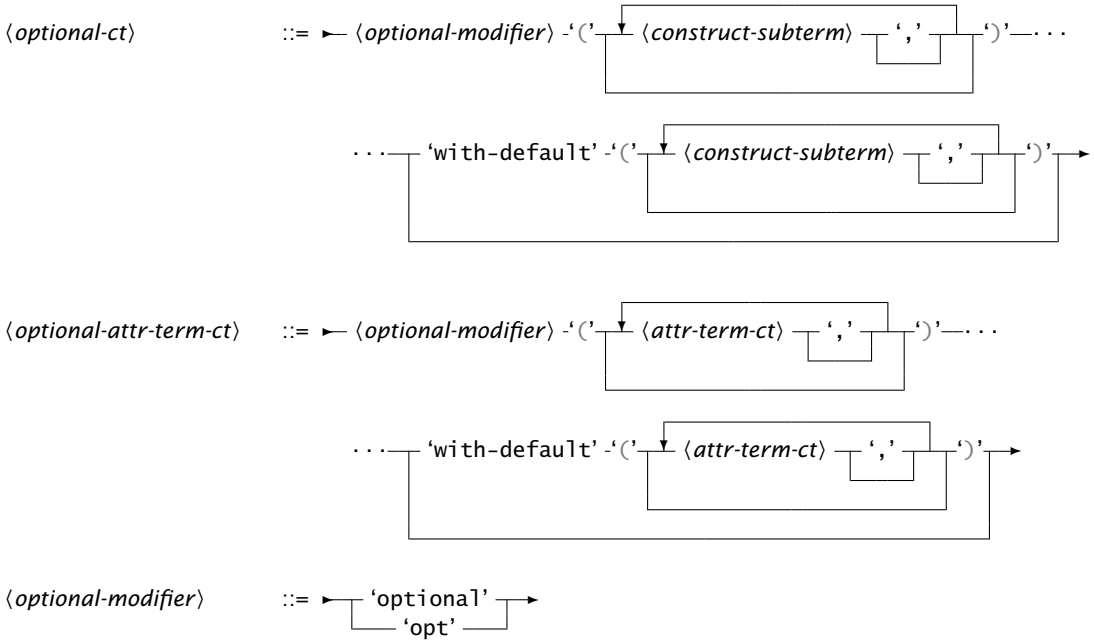
3. **Default value:** Finally, an optional modifier also specifies a default value in the form of another list of construct terms. The default value is used in the result for a substitution σ if for one of the optional variables no binding exist in σ. Note, that the default value *must not contain any of the optional variables*, as they may have no binding.

Notice how an optional construct term resembles a conditional expression of the form **`if`** ⟨*condition*⟩ **`then`** ⟨*modified construct terms*⟩ **`else`** ⟨*default construct terms*⟩.
The condition is, however, always fixed to the existence for bindings for all optional variables in a substitution (cf. Issue 4 for a discussion on general conditional expressions in Xcerpt construct terms).

5.5.1 Textual Term Syntax

Optional modifiers follow very much the syntax of grouping modifiers: the in-scope construct terms follow the **`optional`** keyword (or its shorthand **`opt`**) enclosed in parentheses. The parentheses may be omitted if the scope is a single construct term, as usual. The list of in-scope terms is followed by an optional specification of the default terms, i.e., those terms that are used in the construction, if for one of the optional variables no binding exists. The specification of the default terms is preluded by the **`with-default`** keyword and, as usual, enclosed in parentheses that may be omitted if there is a single default term.

The optional variables can *only* be specified implicitly (cf. Issue 4), i.e., all free variables inside the **`optional`** modifier form the list of optional variables for which bindings must exist so that the in-scope terms are constructed.



5.5.2 XML-style Term Syntax

Again there are no deviations in the productions and non-terminals for the XML-style term syntax. A full grammar is given in Appendix B.3.

5.5.3 Pure XML Syntax

Optional modifiers follow the general syntax for modifiers as introduced in Section 5.4. As in the term syntax, optional variables con only be specified implicitly.

5.6 Instantiating a Construct Term

Summarizing, a construct term C can be "instantiated" by a substitution multi-set B. If each substitution in the substitution multi-set maps all non-optional variables occurring in the construct term to data terms, the result of the instantiation is a data term. The details of the instantiation are described in [38]. The instantiate function shown in Algorithm 5.1 provides an instantiation of a construct term, if called with parameters C, B, the *free* variables in the construct term, i.e., all variables that occur (also) outside of any grouping or optional term, and the empty relation $\emptyset$ as last parameter (i.e., an equivalence relation such that each binding tuple is equivalent to itself only).

In other words, the resulting data terms are obtained by

1. replacing all *free* variables in the construct term, i.e., all variables that occur (also) outside of any grouping or optional modifier, by bindings from each tuple. If there are no free variable a single resulting data term is constructed.
2. replacing each grouping modifier by repeating the in-scope construct terms for each group of grouping variables, each time replacing all occurrences of the grouping variables by the particular group's bindings. This is done recursively for all grouping modifiers.
3. replacing each optional modifier by the in-scope construct terms, if there exists a combination of bindings for all optional variables.

Example 5.1 (Construct Terms). The following substitutions for the variables `Author`, `Title`, and `Publication` are given as result of a query:

Author	Title	Publication
"Cicero"	"Data Processing ..."	**null**
"Cicero"	"Space and ..."	journal[2]
"Antonius"	"Advancements ..."	**null**
"Antonius"	"Efficient Manage..."	proceedings[29]
"Tiro"	"Space and ..."	journal[2]

Notice, that `Publication` is an optional variable.

Then the following construct term in XML-style term syntax

```
<result>
  all <author> var Author </author>
</result>
```

```
fun instantiate(C: construct term list, B: substitution multi-set,
                V: variables, ~: equivalence relation)
  R ← initially empty list of resulting construct terms
  for all t in π~_V(B) do
    for all C ∈ C do
      B⁻ ← B\π_V(B)
      replace V in C by bindings for V in t
      for all G such that G is a grouping modifier directly in C do
        T_G ← the list of modified construct terms of G
        V_G ← grouping variables in G
        ~_G ← equivalence relation used in G
        replace G by instantiate(T_G, B⁻, V_G, ~_G)
      end for
      for all O such that O is an optional modifier directly in C do
        V_O ← optional variables in O
        if ∀ν ∈ V_O : ∃b ∈ B : b is a binding for ν then
          T_+ ← the list of modified construct terms of O
          replace O by sequence of instantiate(T_+, B, V, ~)
        else
          T_− ← the list of default construct terms of O
          replace O by sequence of instantiate(T_−, B, V, ~)
        end if
      end for
      R ← append C to R
    end for
  end for
  return R
end fun
```

Algorithm 5.1: Instantiating a Construct Term (with $\pi^{\sim}$ understood as the projection for tuples using $\sim$ to remove duplicates and "directly in a term T" understood as "occurs in the scope of T but not within the scope of a nested grouping or optional construct term)

results in the single data term

```
<result>
  <author>"Cicero"</author>
  <author>"Antonius"</author>
  <author>"Tiro"</author>
</result>
```

Notice, how Xcerpt defaults to grouping by structural equivalence and thus treats the two substitutions with author “Cicero” as one group, constructing only a single result data term for them.

If we add `Title` as free variable in the scope of the grouping modifier, the grouping variables and thus the groups change:

```
result()[
  all author()[
       var Author
       title()[ var Title ]
     ]
]
```

Leading to the result:

```
result()[
  author()[ "Cicero"
    title()[ "Data Processing ..." ] ]
  author()[ "Cicero"
    title()[ "Space and ..." ] ]
  author()[ "Antonius"
    title()[ "Advancements ..." ] ]
  author()[ "Antonius"
    title()[ "Efficient Manage..." ] ]
  author()[ "Tiro"
    title()[ "Space and ..." ] ]
]
```

Now the substitutions for author and title are both considered for forming a group, leading to more groups!

Nesting grouping modifiers also affects the free variables, e.g., in the following construct term `Title` is no longer free for the out **`all`** only for the inner.

```
result()[
  all author()[
       var Author
       all title()[ var Title ]
     ]
]
```

Thus the result on the sample substitutions is:

```
result()[
  author()[ "Cicero"
    title()[ "Data Processing ..." ]
    title()[ "Space and ..." ]
  ]
  author()[ "Antonius"
    title()[ "Advancements ..." ]
    title()[ "Efficient Manage..." ]
  ]
  author()[ "Tiro"
```

```
    title()[ "Space and ..." ] ]
]
```

Combining grouping an optional modifiers can lead to surprisingly expressive constructs:

```
result()[
  all author()[
     var Author
     all (title()[ var Title ]
         optional var Publication
             with-default standalone()[ ])
    ]
]
```

Results in the following data term:

```
result()[
  author()[ "Cicero"
    title()[ "Data Processing ..." ]
    standalone()[ ]
  ]
  author()[ "Cicero"
    title()[ "Space and ..." ]
    journal.adm @ journal() [ ... ]
  ]
  author()[ "Antonius"
    title()[ "Advancements ..." ]
    standalone()[ ]
  ]
  author()[ "Antonius"
    title()[ "Efficient Manage..." ]
    conf.dmmc @ proceedings()[ ... ]
  ]
  author()[ "Tiro"
    title()[ "Space and ..." ]
    journal.adm @ journal() [ ... ]
  ]
]
```

Chapter 6

How to specify queries? Part 2: Selection

6.1 Specifying Query Patterns: Query Terms

As introduced above, query terms are the second part of expressing the derivation of new data in Xcerpt: where construct term dictate the shape of the new data, query terms specify (possibly incomplete) patterns for data that is to be found, e.g., in Web resources such as XML pages or RDF resource descriptions. As construct terms, query terms enrich basic data terms by variables, but here variables serve to identify data that is to be extracted by the query in form of variable bindings.

Query terms are "matched" with data or construct terms by a non-standard unification method called *simulation unification* that is based on a relation called *simulation* (for details see [11]). In contrast to Robinson's unification (as e.g.used in Prolog), simulation unification is capable of determining substitutions also for incomplete and unordered query terms. Since incompleteness usually allows many different alternative bindings for the variables, the result of simulation unification is not only a single substitution, but a (finite) *multi-set of substitutions*, each of which yielding ground instances of the unified terms such that the one ground term matches with the other.

§1 Query term

Query terms specify *structure and values of data that is to be matched* and which parts of the matched data are to be extracted. In that, they are comparable to clause bodies in Datalog or **`FROM`** and **`WHERE`** clauses in SQL.

Query terms differ more notably from data terms than construct terms do, as they add additional features beyond variables that are essential to express patterns for data, when the data may vary or only limited knowledge about the (shape of the) data is available: In detail, query terms deviate from basic data terms in essentially three aspects (cf. Figure 6.1): the addition of variables, the support for incomplete patterns, and the use of term formulas to express conjunctions, disjunctions, and negations.

To better understand these extensions, an intuition of the answer notion in Xcerpt is needed. The questions, which data and construct terms match with a query term, and what the

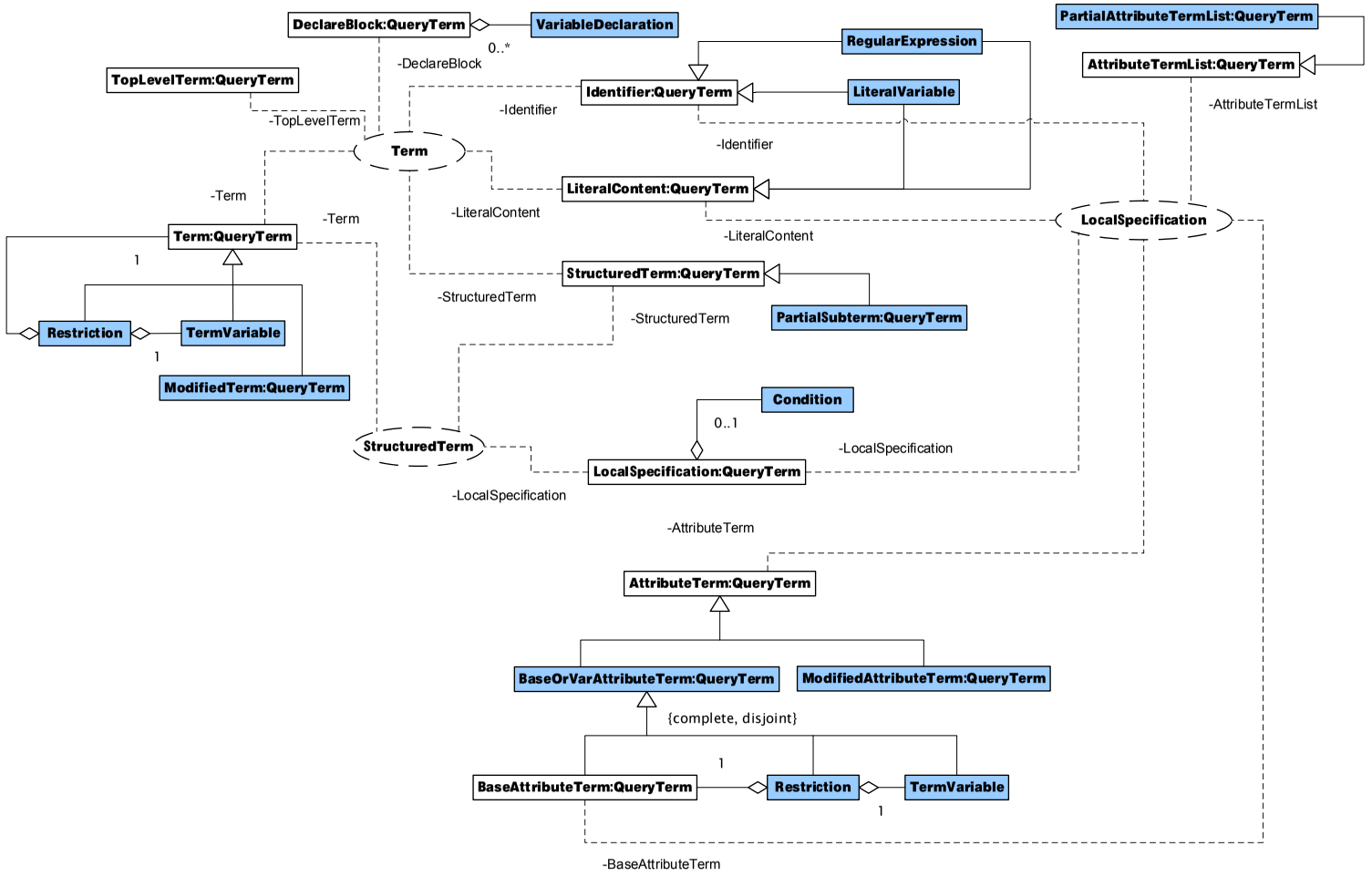


Figure 6.1: UML Model for Query Terms using Parameterized Collaborations

answer (i.e., the substitution multi-set) for a query term is, are formally addressed in [37, 38]. At the root of Xcerpt's answer notion stands an extended form of rooted graph simulation (cf. [34, 27] and [24, 20] for more recent work on efficient algorithms for computing simulation and bisimulation). This extension of the classical notion is necessary to accommodate incomplete patterns as discussed below in Section 6.3.

Intuitively, a query term without any of the extensions discussed in the following matches only with a data term that has *exactly the same shape modulo reordering of direct sub-terms in unordered structured terms and of attributes in any terms*. In the following it is noted, how each of the extensions affect the matching of query terms, but the full details are left to [38].

6.1.1 Textual Term Syntax

The following grammar defines the basic non-terminals for query terms. Notice, the added non-terminal ⟨*modified-qt*⟩. It represents query terms that are possibly modified by varibles or operators discussed in the remainder of this chapter.

Declaration blocks in query terms may also contain variable declarations the details of which are discussed in the next section.

⟨*query-term*⟩ ::= ⟨*modified-qt*⟩ | ⟨*term-level-declare-qt*⟩

⟨*term-level-declare-qt*⟩ ::= 'declare' – ⟨*ns-decl-qt*⟩ – ⟨*variable-decl-qt*⟩ ··· '(' ⟨*query-term*⟩ ',' ')'

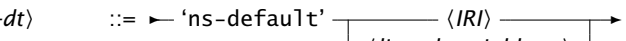
⟨*modified-qt*⟩ ::= ⟨*variable-term-qt*⟩ | ⟨*location-modified-qt*⟩ | ⟨*occurrence-modified-qt*⟩ | ⟨*selection-modified-qt*⟩

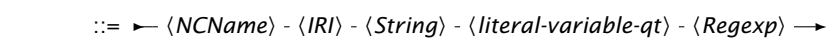
⟨*base-term-qt*⟩ ::= ⟨*reference-qt*⟩ | ⟨*content-qt*⟩ | ⟨*structured-qt*⟩

⟨*reference-qt*⟩ ::= '^' – ⟨*identifier-qt*⟩

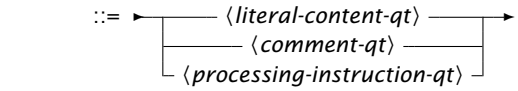
⟨*ns-decl-qt*⟩ ::= ⟨*ns-prefix-decl-qt*⟩ ',' | ⟨*ns-default-decl-qt*⟩ ',' ··· ··· ⟨*ns-prefix-decl-qt*⟩ ','

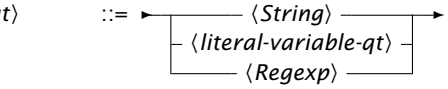
⟨*ns-prefix-decl-qt*⟩ ::= 'ns-prefix' – ⟨*identifier-qt*⟩ – '=' ⟨*IRI*⟩ | ⟨*literal-variable-qt*⟩

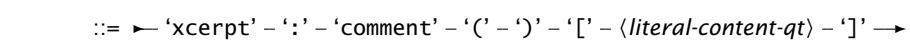
⟨*ns-default-decl-dt*⟩ ::= 

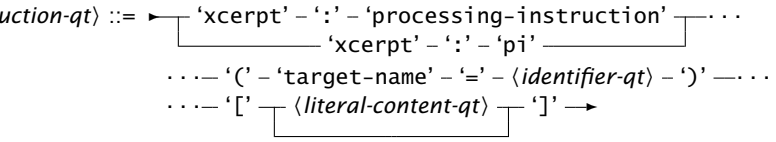
⟨*identifier-qt*⟩ ::= 

Content Query Terms Though the productions for content query terms remain unchanged, the introduction of variables into literal content and identifiers in the next sections indirectly also affect content query terms.

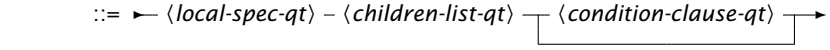
⟨*content-qt*⟩ ::= 

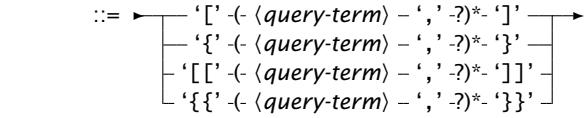
⟨*literal-content-qt*⟩ ::= 

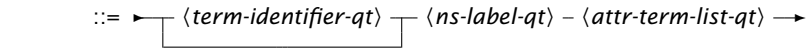
⟨*comment-qt*⟩ ::= 

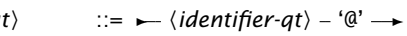
⟨*processing-instruction-qt*⟩ ::= 

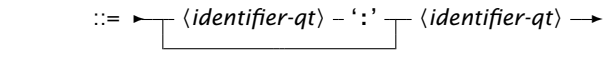
Structured Query Terms Aside of the introduction of variables and their new found ability to occur inside modifiers, structured query terms are nearly identical to structured data terms. Additionally, query terms may be incomplete indicated by double braces or brackets enclosing the list of children. Also query terms can be annotated with condition clauses as described in the next section.

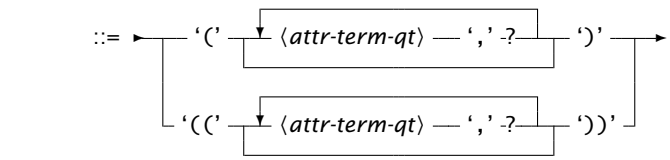
⟨*structured-qt*⟩ ::= 

⟨*children-list-qt*⟩ ::= 

⟨*local-spec-qt*⟩ ::= 

⟨*term-identifier-qt*⟩ ::= 

⟨*ns-label-qt*⟩ ::= 

⟨*attr-term-list-qt*⟩ ::= 

⟨*attr-term-qt*⟩ ::= ⟨*modified-attr-term-qt*⟩

⟨*modified-attr-term-qt*⟩ ::= ⟨*base-attr-term-qt*⟩ | ⟨*variable-attr-term-qt*⟩ | ⟨*occurrence-modified-attr-term-qt*⟩ | ⟨*selection-modified-attr-term-qt*⟩

⟨*base-attr-term-qt*⟩ ::= ⟨*ns-label-qt*⟩ – '=' – ⟨*literal-content-qt*⟩

6.1.2 XML-style Term Syntax

Once more, the XML-style term syntax is closely aligned with the non-XML term syntax, but differs in the representation of comments, processing-instructions, and properties of structured terms. Here, incomplete term specifications are indicated with '`{partial}`' (and complete or total terms specifications with '`{total}`'). The same applies for incomplete attribute term lists. As in data and construct terms, query terms are syntactically similar to XML elements, but may contain Xcerpt constructs additionally.

⟨*comment-qt*⟩ ::= '<!-' – ⟨*literal-content-qt*⟩ – '->'

⟨*processing-instruction-qt*⟩ ::= '<?' – ⟨*identifier-qt*⟩ – ⟨*literal-content-qt*⟩ – '?>'

⟨*structured-qt*⟩ ::= ··· '<' – ⟨*local-spec-qt*⟩ – ⟨*properties-qt*⟩ ('>' – ⟨*children-list-qt*⟩ ('</>' | '<' – ⟨*ns-label-qt*⟩ – '>') | '/>') ··· | ⟨*condition-clause-qt*⟩ ···

⟨*properties-qt*⟩ ::= ··· ('{' – 'ordered' – '}' | '{' – 'unordered' – '}') ('{' – 'total' – '}' | '{' – 'partial' – '}') | ('{' – 'total attributes' – '}' | '{' – 'partial attributes' – '}') ···

⟨*children-list-qt*⟩ ::= ⟨*query-term*⟩

⟨*attr-term-list-qt*⟩ ::= ⟨*attr-term-qt*⟩

6.1.3 Pure XML Syntax

Once more, the pure XML syntax for query terms relies heavily on the parameterizable Relax NG grammar for terms introduced in Section 4.1. Evidently, query terms can become the most complex of the three term kinds in Xcerpt, cf.Figure 6.2. As construct terms they add variables to data terms. But they also provide means for expressing incompleteness: partial terms, **`descendant`** and **`position`** location modifiers, etc.

Query terms deviate from the basic term case in (a) extended top-level terms, (b) the introduction of modified-terms, (c) the use of condition formulas attached to each term, and (d) the use of variables and regular-expressions in identifiers and literal content.

```
## A POSIX.1 regular expression annotated with variables may occur in
## query terms at the position of identifiers or literal content.
regular-expression =
  element regexp {
    attribute value { text }
  }

query-term =
  grammar {

    include "term.rnc" {
      # Redefine the top-level term for query terms: add variables to
      # declare blocks and allow optional, descendant, variable restriction.
      # Add document specifications
      # Add query term formula

      ## A term that may occur at top-level. Slightly more
      ## restricted than a basic term.
      top-level-term.class =
        optional-top-level-term
        | term-formula
        | document-specification
        | grammar {
            include "declare-block.rnc" {
              content = parent top-level-term.class*
            }
          }
      # Redefine terms: only modified terms, which can in fact be
      # unmodified :-) Term-level declare blocks may also contain variable
      # declarations

      ## A generic Xcerpt term. Variants are data, construct, and query terms.
      term.class = modified-term | term-level-declare

      ## A declaration block on term level allows possibly (in data and construct terms) only namespace
          declarations.
      term-level-declare =
        grammar {
          include "declare-block.rnc" {
            content = parent term.class*
          }
        }

      ## An attribute term is an attribute possibly modified with respect to location, modality, and selection.
      attribute-term.class = modified-attr-term
      # Allow conditions on arbitrary query terms
      term-condition = condition-clause

    }

    # Add variables and regular expressions to identifiers and literal
    # content
    identifier.class |= variable | parent regular-expression
    literal-content.class |= variable | parent regular-expression

    ## Variables for query terms.
    variable =
      element variable {
        attribute anonymous { "true" }
```

```
        | attribute name { xsd:NCName }
60    }

62    # #1# TOP-LEVEL QUERY TERM
      # see below
64
      # #2# CONDITION CLAUSES
66    # see below

68    # #3# MODIFIED TERMS AND ATTRIBUTE TERMS
      # see below
70
  }
```

6.2 Variables in Query Terms

Variables in query terms are used for three purposes: (a) to specify *which parts of a matched (construct or data) term are "selected"* by the query and can be used in the corresponding construct term, (b) to specify *joins*, i.e., multiple occurrences of the same data term or literal value (usually unknown at time of query authoring), and (c) to specify arithmetic or other *conditions* involving (literal) values of variables.

Like in construct terms variables may be used in query terms in place of (a) (structured or attribute) terms or in place of (b) identifiers and literal content. In either case, a variable matches (unless further restricted as discussed below) any sub-term or literal that may occur at that position, i.e., regardless of the shape of the sub-term or literal.

Variables are *named* so that they can be referred to in other parts of the query term (forming a join) or in the corresponding construct term. Xcerpt allows in addition to named variables also contain **anonymous variables** (like in Prolog). As unrestricted named variables, an anonymous variables matches arbitrary terms or literals that may occur at the position of the variable. However, bindings for anonymous variables are not recorded and different occurrences of the same anonymous variables are treated like different named variables. Thus, anonymous variables can neither be used for joins, nor be restricted through variable restrictions or conditions, nor occur in construct terms. Their sole purpose is to act as a wildcard construct.

Additionally, query terms may contain so-called **variable restrictions**, where a variable does not just replace some sub-term (and thus is bound to all sub-terms in a matching data term that can occur at that point), but the sub-terms that may be bound to the variable are further restricted by specifying a arbitrary query term.

§2 Variable Restrictions

A variable restriction places a *constraint on the structure* of (data or construct) terms that can be bound to the restricted variable by specifying the possible shapes of such (data or construct) terms as a query term.

Variable restrictions may only occur in place of structured and attribute query terms, not in place of identifiers or literal content (cf. Issue 12). Figure 6.3 shows variable restrictions in the context of basic query terms: Each variable restriction restricts one (term) variable to a (basic) query term (the scope of that variable restriction). Variable restrictions for attribute terms are analogous.

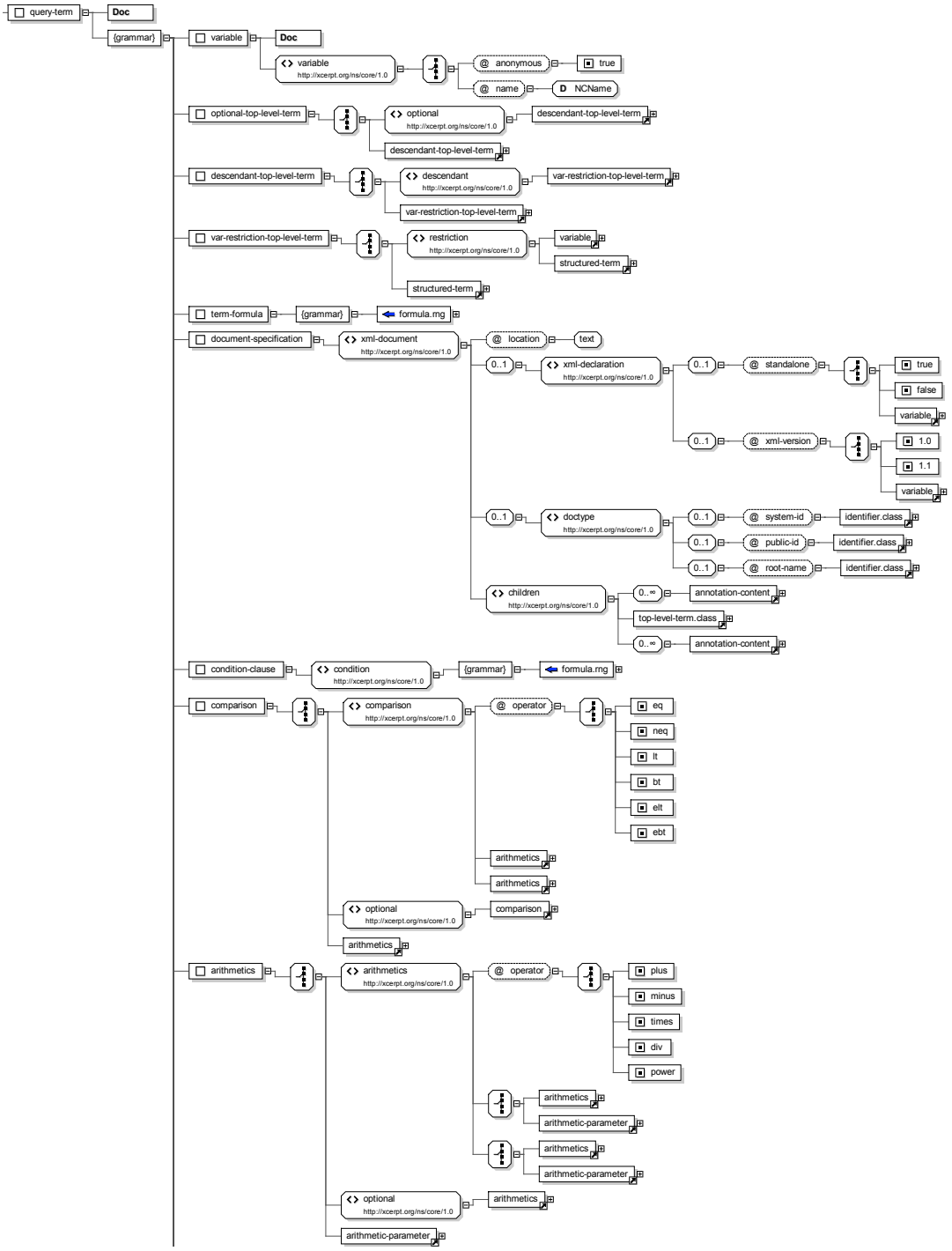


Figure 6.2: Relax NG Grammar for Query Terms (Excerpt)

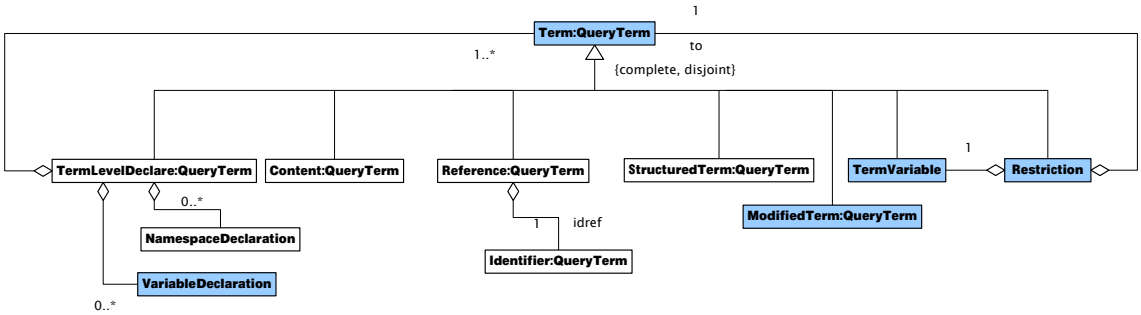


Figure 6.3: UML Model for (basic) Query Terms

Another means to restrict the bindings of a variable are **conditions**. Where variable restrictions restrict the terms a variable can be bound to, conditions restrict the *values*:

§3 Conditions

Conditions restrict the possible *(literal) values* a variable can be bound to and thus are mostly used to restrict literal variables, i.e., variables that occur in place of identifiers or literal content.

Conditions consist in arbitrary expressions (though currently, only arithmetic and comparison expressions on (floating point) numbers are defined, cf. Section 8.2.3) formed with identifiers or literal content as atoms. Also conditions may use boolean connectives to form condition formulas. A more detailed description of conditions (and functions in general) is under development, cf. Issue 8.

Rather than restricting the terms or literal values that a variable can be bound to, term variables may also be *bound to only parts of a matched sub-term*:

§4 `except` Binding Modifier

A binding modifiers changes the binding a variable without affecting the match of a query. The only kind of binding modifier in Xcerpt is the **`except` binding modifier**. Using **`except`** a part of a sub-term can be omitted from the bindings.

The *scope* of an **`except`** modifier is a single (possibly modified basic or attribute, resp.) query term as seen in Figures 6.4 and 6.5 (cf. Issue 9).

Since **`except`** changes variable bindings it is only useful in *the scope of a variable restriction*. Occurrences outside of any variable restriction are ignored (cf. Issue 9).

Notice, that **`except`** does not change the original matched term, but only the variable bindings. It also does not affect the matching of a query term: The query term obtained by replacing all **`except`**'s in a query term by their in-scope query terms matches the same data or construct terms as the original one. The only differences is that some variable bindings might have certain parts removed.

Variables may be *declared* in term-level declare blocks just like namespaces.

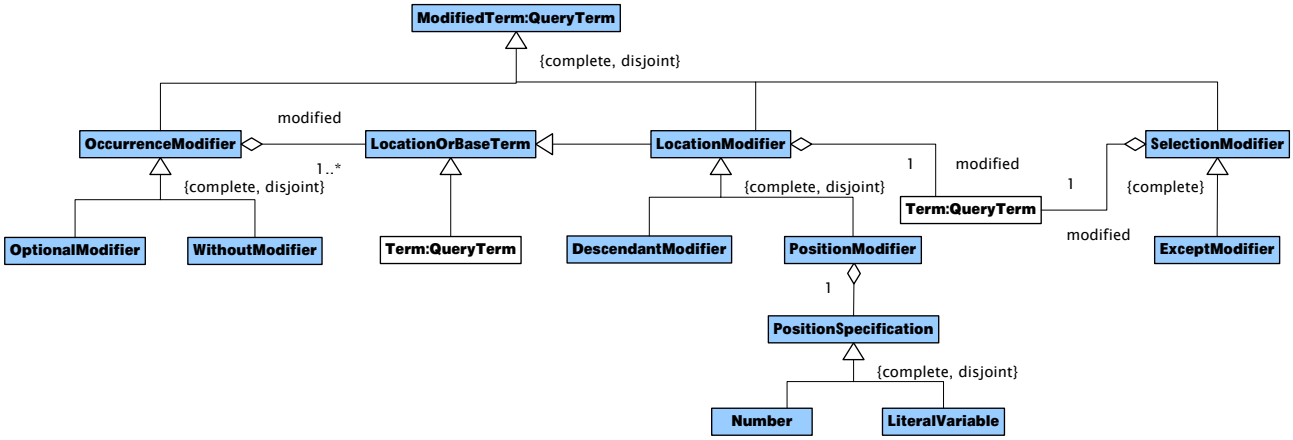


Figure 6.4: UML Model for Modified Structured Query Terms

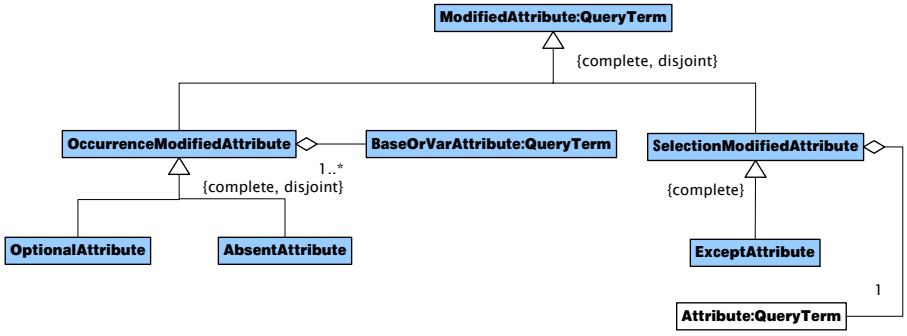


Figure 6.5: UML Model for Modified Attribute Query Terms

§5 Variable Declarations

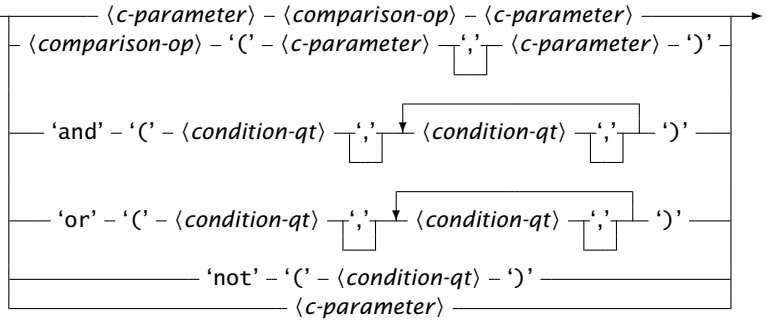
A **variable declaration** is, at the time of writing, only reserving a certain identifier for use as a variable in the scope of the declaration block.

However, it is envisioned that in the same way variables might be typed, e.g., to restrict a certain variable to literal values or to structured values only (cf. Issue 15).

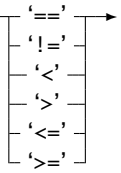
6.2.1 Textual Term Syntax

⟨*condition-clause-qt*⟩ ::= 'where' '(' ⟨*condition-qt*⟩ ')'

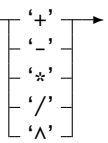
⟨*condition-qt*⟩ ::=



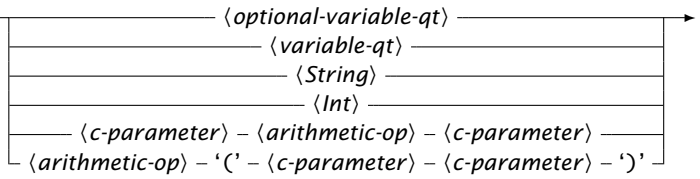
⟨*condition-op*⟩ ::=



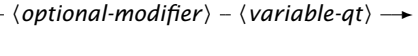
⟨*arithmetic-op*⟩ ::=



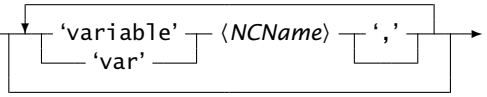
⟨*c-parameter*⟩ ::=



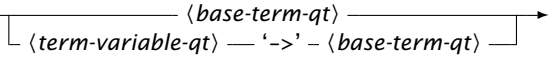
⟨*optional-variable-qt*⟩ ::=

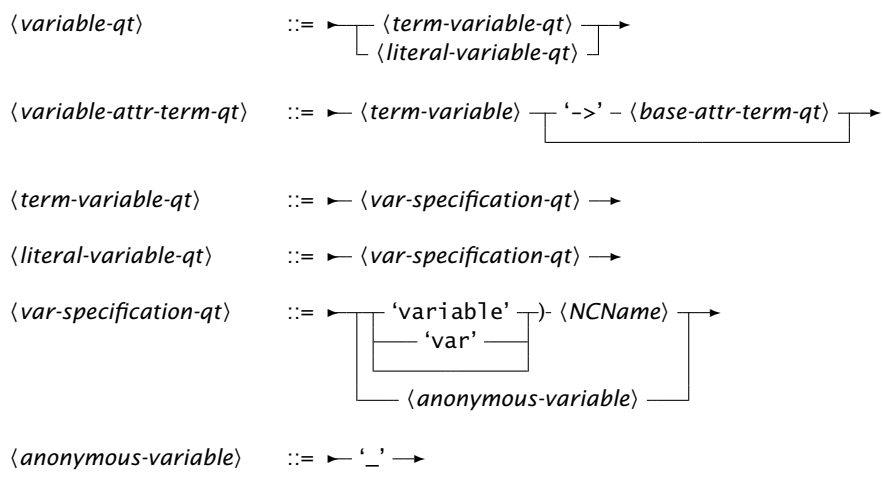


⟨*variable-decl-qt*⟩ ::=



⟨*variable-term-qt*⟩ ::=





6.2.2 XML-style Term Syntax

Once more, the XML-style term syntax uses productions identical to the ones for the non-XML term syntax. The full grammar is given in Appendix B.4.

6.2.3 Pure XML Syntax

The following parameterized grammar for formulas is used not only for defining condition clauses at term-level, but also for term formulas only occuring at top-level in query terms.

```
default namespace = "http://xcerpt.org/ns/core/1.0"
namespace a = "http://relaxng.org/ns/compatibility/annotations/1.0"

start = formula
formula =
  element and { formula, formula+, condition? }
  | element or { formula, formula+, condition? }
  | element not { formula }
  | content
condition = empty
content = empty
```

Condition clauses consist in such formulas, where the content can be comparisons and arithmetic expressions over variables.

```
    # #2# CONDITION CLAUSES ##
  condition-clause =
    element condition {
      grammar {
        include "formula.rnc" {
          content = parent comparison*
        }
      }
    }
  comparison =
    element comparison {
      attribute operator {
        "eq" | "neq" | "lt" | "bt" | "elt" | "ebt"
      },
```

```
15      arithmetics,
        arithmetics
17    }
      | element optional { comparison }
19    | arithmetics
    arithmetics =
21    element arithmetics {
        attribute operator {
23        "plus" | "minus" | "times" | "div" | "power"
        },
25      (arithmetics | arithmetic-parameter),
        (arithmetics | arithmetic-parameter)
27    }
      | element optional { arithmetics }
29    | arithmetic-parameter
    arithmetic-parameter =
31    variable
      | element value { text }
```

6.3 Incomplete Patterns

As discussed above, query terms are meant to be examples or patterns of the sought-for data. So far, however, query terms always have to specify all parts of matched data, even if only a small part is sufficient to distinguish relevant from irrelevant data. Obviously, this is unacceptable for many queries. Therefore, Xcerpt introduces a number of concepts to allow query terms to be *incomplete patterns* of the sought-for data, that may specify only what is needed to distinguish relevant from irrelevant data. In contrast to many other query languages (such as XQuery and SQL) that assume that a query specifies only parts of the sought-for data and make it difficult to specify queries where no additional data may occur, Xcerpt patterns make it *obvious* where a query term is incomplete and where not. This is a property that is particularly welcome in the context of semi-structured data as here the schema of the data is often unknown or variable, allowing, e.g., optional or repeated children.

Query terms (i.e., patterns) can be incomplete with respect to

1. **breadth**, i.e., only a subset of the actual children of a term is specified.

§6 Partial Terms

A term is called *partial*, if only some of the children of the term are specified. When matching partial terms, additional children may occur without affecting the match. However, all specified children must occur and, in case of ordered terms, must occur in the same order.

If a term is not *partial*, it is called *total* as discussed above. Notice, that neither data nor construct terms may contain partial terms, as data is always presumed to be "complete" (cf. Issue 35).

Partial terms are obviously essential for dealing with semi-structured data, where the schema of the data may allow for repetition or omissions of data (both in breadth and depth). Thus, queries can not specify total (or complete) patterns for the data.

However, partial terms also introduce a number of new challenges in a pattern language such as Xcerpt—that do not occur, e.g., in logic programming languages such as Prolog, where term arity and children order of two matching terms are always the same.

First, assume an *ordered* query term q that specifies only a partial list of children. Then the position (among its siblings) of a match m' for a child of m may (and will in most cases) differ from the position of m among its siblings (i.e., among q's children). However, in many cases access to the (sibling) position is needed, e.g., to obtain the first child or immediate following sibling of a matched term. Therefore, Xcerpt provides access to the *sibling position* through the position modifier:

§7 `Position` Modifier

The **`position` modifier** allows the specification of the position of a child among its siblings, i.e., in the list of children of its parent. The *position* may be specified as an arithmetic expression composed of the usual arithmetic operators for natural numbers and natural numbers as well as (literal) variables as atomic expressions. The *scope* of a **`position`** modifier is a single term.

Obviously, a **`position`** modifier can only occur inside other (structured) terms and not at the top-level of a query term.

Using variables one can (a) query the position, (b) express positional joins, e.g., to find immediate siblings, and (c) correlate data and position. Variables occurring in position specifications may only be bond to natural numbers, otherwise a (run-time) exception occurs (cf.Issue 15). The following example shows the use of positional variables to find the immediate two following siblings of a term:

Example 6.1. immediate following two siblings

Second, though one may not be able to specify how all of the children of a sought-for term ought to be shaped like, one might be able to specify how they ought *not* to be shaped. Again, this makes sense only in a partial term, as in a total term the shape of all children must be explicitly stated. Xcerpt uses the **`without`** modifier to express this *subterm negation*:

§8 `Without` Modifier

The **`without` modifier** expresses that *not all sub-terms in its scope* may match with a sub-term of a match for its parent term. The scope of a **`without`** modifier is a list of one or more sub-terms.

Sub-terms in the scope of a **`without`** modifier are often referred to as *negative*, those outside as *positive*.

Notice, that the definition does not specify that *none* of the sub-terms may occur in a match for the parent: If the scope of a **`without`** modifier is a list of sub-terms, than the parent term t matches another term t', if there is a mapping of the positive sub-terms of t to t', that can not be extended to also cover all negative sub-terms. Thus, subterm negation is *existential*.

Example 6.2. examples with a list and a single without

Like **`position`** modifiers, **`without`** modifiers may only occur in sub-terms. Note, also that Xcerpt places some limitations on variables used in **`without`** modifiers, cf.Section 7.2.

Currently, **`without`** modifiers may not be either arbitrary siblings of each other in an unordered parent term or immediate siblings of each other in an ordered parent term (cf. Issue 6).

Figure 6.6: UML Model for Qualified Descendant

2. **depth**: Semi-structured data may not only vary in the number, order, and repetition of children, but also in how elements are nested. E.g., in (X)HTML most inline elements such as `em` may occur in most block-level elements such as `p` or `div`, but may also be nested inside each other. Thus selecting all `em` elements in an (X)HTML document in a pattern requires a means to specify patterns that are incomplete in depth, i.e., that contain sub-terms that are not direct sub-terms of their parent but stand in another structural relation to it, e.g., occurring at any depth under their parent or occurring at depth 5 under their parent.

   To express such incompleteness in depth Xcerpt provides the **`descendant`** modifier, similar in its basic form to the **`descendant`** axis in XPath. In contrast to XPath (and thus XSLT and XQuery), Xcerpt also provides a more expressive variant of the **`descendant`** modifier that allows direct expression of constraints such as "occurs at depth 5 under its parent" or "occurs at any depth under its parent but with only `div` elements in between its parent and itself". The latter variant of the **`descendant`** modifier is referred to as *qualified*, the basic case as *unqualified*.

§9 (Unqualified) `Descendant` Modifier

The unqualified variant of the **`descendant` modifier** specifies that the *single sub-term* in its scope may occur at any depth under the parent term (rather than as an immediate child).

Notice, that the **`position`** and **`descendant`** modifier can not be mixed, as the former to the position among the immediate children of the parent term, and the latter specifies that the sub-term may also be nested more deeply inside the parent term.

In contrast to the **`position`** modifier the **`descendant`** modifier may occur at top-level, thus specifying that the contained term may occur at any level in the document.

§10 Qualified `Descendant` Modifier

The qualified variant of the **`descendant` modifier** specifies a more involved relation between the parent term and the single in-scope sub-term: The in-scope sub-term occurs inside the parent term, but the path in between is restricted by a qualifying expression that consists in a selection and a repetition part.

A detail model of the qualified descendant is given in Figure 6.6: The *selection* part is a sequence of (one or more) element label and optional attribute term specifications, both possibly containing variables. The *repetition* part is an interval $[i, j]$ with $i <= j$ and $i, j \in \mathbb{N}_0 \cup \{\infty\}$. The interval boundaries may also be literal variables.

Thus, the qualified descendant restricts the in-scope sub-term to matchings that are reached from a match of the parent term via a path that matches the selection part repeated between i and j times.

Notice, that variable occurrences in the selection and repetition part of a qualified descendant are non-binding, i.e., all such variables must be bound in another part of the query term (cf. Section 7.2).

Example 6.3.

3. **optional parts:** One of the most distinguishing features of semi-structured data in contrast to, e.g., relational data aside is the allowance for optional information, i.e., information that occurs in some elements of a certain type but is missing in others of the same type. Though *testing* for the existence or absence of such optional information has been a focus in many semi-structured and XML query languages (most notably structural predicates in XPath), *selecting* of or *construction* based on optional information has been far less closely investigated. Xcerpt provides query authors with a unified concept for handling optional information in the context of testing, selection, and construction, quite in contrast to mainstream XML query languages such as XQuery and XSLT.

   Just like in construct terms, the optional modifier is used in query terms to indicate which parts of a query may be missing without affecting the matching of the remainder of the query.

§11 Optional Modifier

The **optional modifier** in query terms indicates that its in-scope terms may be missing in a matching term, but have to be considered, if they are existent. The scope of an optional modifier is a list of terms.

This way `optional` modifiers serve to bind variables to part of the data that may be absent, but that must be included in the substitutions resulting from a query, if present. Obviously, only bindings for variables that do not occur (positively, cf. Section 7.2) also outside of the scope of any `optional` are effect by the presence or absence of the optional part, as in the other case their bindings are already established by the outside occurrence. Therefore, an `optional` modifier with no such variable in scope does not affect either matching or the resulting substitutions and can thus be safely removed from the query.

4. **order**: As discussed above, data and construct terms may already be distinguished in ordered and unordered terms. Often, however, one might not care about the order in which matches for the sub-terms in a query occur in the data, even if the data itself is ordered. Xcerpt acknowledges this fact by allowing query terms that are unordered to match with ordered terms, but not the other way around. I.e., if the query specifies the order is significant then only data where the order is significant as well can match with that query; if the query however indicates that the order may be ignored, then also data is considered that is ordered, however the sub-terms of the query are matched in any order with the sub-terms of the data.

5. **literal specification**: Finally, like in the relational case, queries often may not be able to specify literal content or identifiers completely, but rather query for data where the literal content or the identifiers falls into some class, specified in Xcerpt by means of POSIX.1 regular expressions enhanced with variable bindings: additionally to using POSIX's numeric backreferences, Xcerpt allows subexpressions to be bound to Xcerpt literal variables. This allows the extraction and insertion of data from the rest of the Xcerpt query into the regular expression.

   As stated, regular expressions may occur anywhere in a query term where literal content or identifiers may occur, except where only natural numbers are allowed, as in repetition and position specifications.

   Notice, that for the "wildcard" regular expression `.*` anonymous variables may be used and are often more convenient.

6.3.1 Textual Term Syntax

⟨*selection-modified-qt*⟩ ::= ⟨*selection-modifier*⟩ – '(' ⟨*modified-qt*⟩ ',' ')'

⟨*selection-modified-attr-term-qt*⟩ ::= ⟨*selection-modifier*⟩ – '(' ⟨*modified-attr-term-qt*⟩ ',' ')'

⟨*selection-modifier*⟩ ::= 'except'

⟨*occurrence-modified-qt*⟩ ::= ⟨*occurrence-modifier*⟩ – '(' ⟨*modified-qt*⟩ ',' ')'

⟨*occurrence-modified-attr-term-qt*⟩ ::= ⟨*occurrence-modifier*⟩ – '(' ⟨*modified-attr-term-qt*⟩ ',' ')' ···

⟨*occurrence-modifier*⟩ ::= ⟨*optional-modifier*⟩ | 'without'

⟨*location-modified-qt*⟩ ::= ⟨*location-modifier*⟩ – '(' ⟨*term-variable-qt*⟩ ',' ')'

⟨*location-modifier*⟩ ::= ⟨*descendant-modifier*⟩ | ⟨*position-modifier*⟩

⟨*descendant-modifier*⟩ ::= 'descendant' | 'desc'

⟨*position-modifier*⟩ ::= 'position' ⟨*number-qt*⟩ | 'pos'

⟨*number-qt*⟩ ::= ⟨*Int*⟩ | ⟨*literal-variable-ct*⟩

6.3.2 XML-style Term Syntax

Once more, the XML-style term syntax uses productions identical to the ones for the non-XML term syntax. The full grammar is given in Appendix B.4.

6.3.3 Pure XML Syntax

Like in the term syntax, the XML syntax enforces a hierarchy of modified terms with occurrence and selection modified terms at the top followed by occurrence modified terms, variable restrictions and finally base terms.

```
# #3# MODIFIED TERMS ##
modified-term =
  variable-term | location-term | occurrence-term | selection-term
base-term = reference | content-term | structured-term
variable-term =
  base-term
  | variable
  | element restriction { variable, base-term }
location-term =
  element descendant { variable-term }
  | element position {
     element number { variable | xsd:int },
     variable-term
   }
selection-term = element except { modified-term }
occurrence-term =
  element without { modified-term }
  | element optional { modified-term }
# #4# MODIFIED ATTRIBUTE TERMS ##
modified-attr-term =
  base-attribute,
  variable-attr-term,
  occurrence-modified-attr-term,
  selection-modified-attr-term
variable-attr-term =
  variable
  | element restriction { variable, base-attribute }
occurrence-modified-attr-term =
  element without { modified-attr-term }
  | element optional { modified-attr-term }
selection-modified-attr-term = element except { modified-attr-term }
```

6.4 Top-level Query Terms

As mentioned in the discussion of some of the query term modifiers above, certain modifiers are only allowed at sub-term level, but not at the top-level. On the other hand, there are some constructs that may only occur in top-level query terms, viz. term formulas and document specifications. Figure 6.7 shows a detailed model of top-level query terms. Notice that from all modifiers in query terms only **`optional`** and **`descendant`** modifiers are allowed at top-level.

6.4.1 Term Formulas

(Top-level) query terms can be connected by the usual boolean connectives to form so-called query *term formulas*.

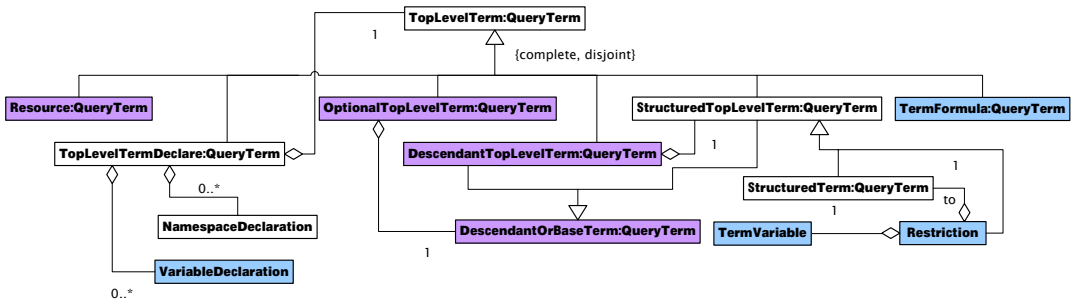


Figure 6.7: UML Model for Top-level Query Terms

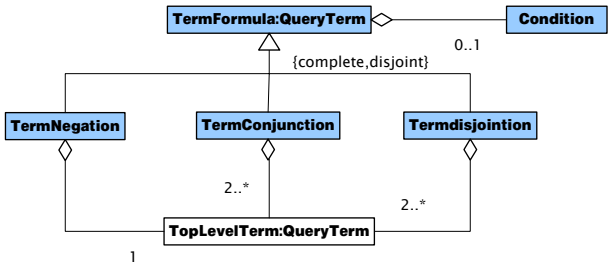


Figure 6.8: UML Model for Query Formula

§12 Term Formulas

A **term formula** is an expression formed from boolean connectives (**and**, **or**, and **not**) over top-level query terms. Intuitively, **or** merely merges the resulting multi-sets of substitutions resulting from the contained queries (similar to **union** in SQL database systems), whereas **and** creates the joins the individual substitutions (if none of the queries contains a negation).

Besides the sub-term negation introduced in Section 6.3 above (**without** modifier), Xcerpt also supports *query negation*, denoted using **not**. The query negation used in Xcerpt is negation as (finite or infinite) failure like in logic programming, i.e., a negated query **not**(Q) succeeds if the query Q fails. Like in negated sub-terms, variables occurring in a negated query do not yield bindings, i.e., they have to appear elsewhere in the query outside the scope of a negation construct (cf.Section 7.2).

Notice, that query negation is universal quantified, i.e., there may be no term that matches the query, whereas sub-term negation filters out those parent terms that contain the negated sub-term, and thus is effectively existential quantified if the parent term is not bound to a variable.

6.4.2 Document Specifications

So far, query terms have not specified what document the data they are matched against comes from. In this case, data and construct terms that are part of the program (or set-up by some other environment specific method) are considered. If data stored in external sources is to

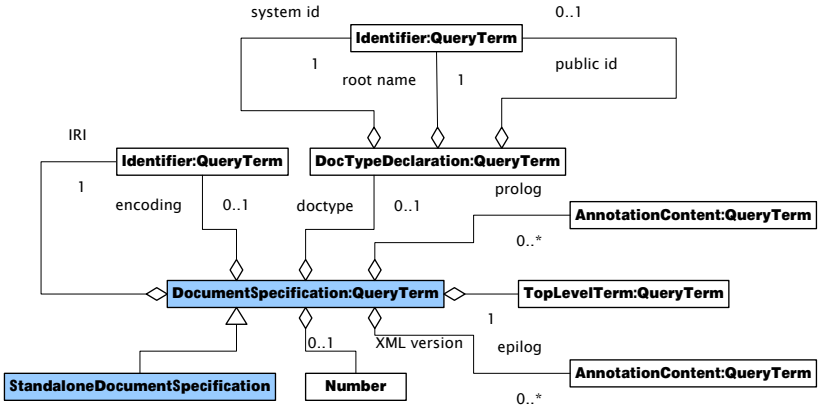


Figure 6.9: UML Model for Document Specifications

be accessed, a *document specification* is needed to specify the needed information about that external data source.

§13 Document Specification

A **document specification** describes an external data source such as an XML document. Typically document specifications contain at least *access parameters*, e.g., the IRI (Internationalized Resource Identifier [21]) of an XML document.

At the time of writing, the only form of document specification are XML document specifications.

§14 XML Document Specification

An **XML document specification** is a document specification to specify access to XML documents. Aside of an IRI identifying the document to be accessed, an XML document specification may contain most of the information present in the document and document type declaration information items from [16]: XML version, standalone status flag, root name (i.e., the tag name of the document element), system, and public identifier of the document type declaration if any, as well as the document element (a top-level term) and two lists of annotation content (i.e., processing instruction or comment terms) for document prolog and epilog.

Note, that there is currently no mechanism in Xcerpt to enable or disable validation in presence of a document type declaration. In absence of such a mechanism, Xcerpt implementations are expected to validate all documents with document type declaration. If such a document is not valid, an error is generated. Thus, if a root name is present, it will always be the same as the label of the document element.

For querying, document specifications are treated just like any other form of data, i.e., one can, e.g., query all documents conforming to a DTD identified by a specific public or system ID.

Document specifications are also used for specifying properties of output documents, cf. Section 7.

6.4.3 Textual Term Syntax

As noted, top-level query terms may, in addition to structured query terms and top-level declare blocks as in data and construct terms, may also be modified through **optional**, **descendant**, or a variable restriction and may be formulas of query terms as well as document specifications, i.e., expression for accessing specific resources on the Web.

⟨*top-level-query-term*⟩ ::= ⟨*top-term-level-declare-qt*⟩ | ⟨*optional-top-level-qt*⟩ | ⟨*term-formula-qt*⟩ | ⟨*document-specification-qt*⟩

⟨*top-term-level-declare-qt*⟩ ::= ··· 'declare' – ⟨*ns-decl-qt*⟩ – ⟨*variable-decl-qt*⟩ → '(' – ⟨*top-level-construct-term*⟩ – ')' ···

⟨*optional-top-level-qt*⟩ ::= ⟨*optional-modifier*⟩ – '(' – ⟨*descendant-top-level-qt*⟩ – ')' | ⟨*descendant-top-level-qt*⟩

⟨*descendant-top-level-qt*⟩ ::= ⟨*descendant-modifier*⟩ – '(' – ⟨*var-restriction-top-level-qt*⟩ – ')' | ⟨*var-restriction-top-level-qt*⟩

⟨*var-restriction-top-level-qt*⟩ ::= ⟨*term-variable*⟩ – '->' – ⟨*structured-term-qt*⟩ | ⟨*structured-term-qt*⟩

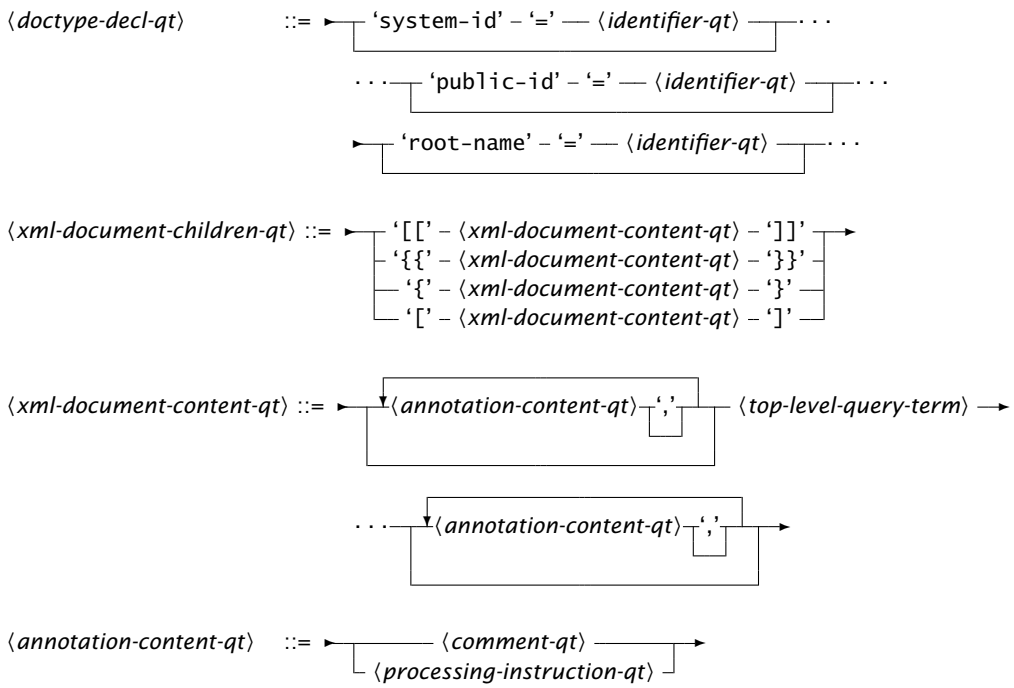
⟨*term-formula-qt*⟩ ::= 'not' – '(' – ⟨*top-level-qt*⟩ – ')' ··· | 'and' / 'or' – '(' – ⟨*top-level-qt*⟩ – ',' – ⟨*top-level-qt*⟩ – ',' – ')' | ··· ⟨*condition-clause-qt*⟩

⟨*document-specification-qt*⟩ ::= 'in' – ⟨*xml-document-specification-qt*⟩ →

⟨*xml-document-specification-qt*⟩ ::= ··· 'xml-document' – '((' – ⟨*location-qt*⟩ | ⟨*xml-decl-qt*⟩ → | ⟨*doctype-decl-qt*⟩ | '))' – ⟨*xml-document-children-qt*⟩ ···

⟨*location-qt*⟩ ::= 'location' – '=' – ⟨*IRI*⟩ -|- ⟨*literal-variable-qt*⟩ →

⟨*xml-decl-qt*⟩ ::= 'standalone' – '=' – 'true' / 'false' / ⟨*literal-variable-qt*⟩ ··· | ··· 'xml-version' – '=' – '1.0' / '1.1' / ⟨*literal-variable-qt*⟩ →



6.4.4 XML-style Term Syntax

Once more, the XML-style term syntax uses productions identical to the ones for the non-XML term syntax. The full grammar is given in Appendix B.4.

6.4.5 Pure XML Syntax

Top-level query terms differ from general top-level terms in the possible use of a **optional** or **descendant** modifier, the possible use of a variable restriction and, most notably, in the addition of term formulas and document specifications, both occuring exclusively at the top-level of query terms. Notice the reference to the generic grammar for formulas using ⟨*optional-top-level-term*⟩ as basic content type.

```
# #1# TOP−LEVEL QUERY TERM ##
optional-top-level-term =
  element optional { descendant-top-level-term }
  | descendant-top-level-term
descendant-top-level-term =
  element descendant { var-restriction-top-level-term }
  | var-restriction-top-level-term
var-restriction-top-level-term =
  element restriction { variable, structured-term }
  | structured-term
term-formula =
  grammar {
    include "formula.rnc" {
      content = parent optional-top-level-term
      condition = parent condition-clause
    }
```

```
    }
18  document-specification =
     element xml-document {
20     attribute location { text },
       element xml-declaration {
22       attribute standalone { "true" | "false" | variable }?,
         attribute xml-version { "1.0" | "1.1" | variable }?
24     }?,
       element doctype {
26       attribute system-id { identifier.class }?,
         attribute public-id { identifier.class }?,
28       attribute root-name { identifier.class }?
       }?,
30     element children {
         annotation-content*, top-level-term.class, annotation-content*
32     }
     }
```

6.5 Summary: Modifiers and Where they Occur

To address the needs of querying and construction, Xcerpt provides quite a number of modifiers that affect the way their in-scope terms are handled.

Most of these modifiers may occur either in construct or in query terms. The single exception from this rule is the **optional** modifier: it marks both parts of construct terms that may or may not occur depending on the result of the query term and parts of query terms that may or may not yield bindings depending on the data the query term is matched against.

Table 6.1 summarizes the eight modifiers (and three boolean connectives for term formulas) and gives at a glance which modifier may occur where and with what scope.

As discussed above, the following additional constraints hold:

1. In construct terms grouping and occurrence (**optional**) modifiers may be arbitrarily mixed. Grouping modifiers on attributes are limited to a single term as scope to avoid the repetition of attributes with the same name.

2. In query terms location modifiers "stick closely to the term modified by them", i.e., occurrence and selection modifiers may contain location modifiers, but not vice versa. Therefore, location modifiers also always affect only a single term.

3. Finally, for top-level query terms only **optional** and **descendant** are allowed (the latter may occur inside the former), each with a single in-scope term.

	Data Terms	Construct Terms		Query Terms		
		Subterms	Attributes	Subterms	Attributes	Top-level
Grouping Modifiers						
`all` (all bindings)	—	$*$	•	—	—	—
`some` (some m bindings)	—	$*$	•	—	—	—
`first` ($n^{\text{th}} - m^{\text{th}}$ bindings)	—	$*$	•	—	—	—
Selection Modifiers						
`except` (omit from binding)	—	—	—	$*$	$*$	—
Occurrence Modifiers						
`optional` (may occur)	—	$*$	$*$	$*$	$*$	•
`without` (must *not* occur)	—	—	—	$*$	$*$	—
Location Modifiers						
`descendant` (at any depth)	—	—	—	•	—	•
`position` (as n^{th} child)	—	—	—	•	—	—
Term formulas						
`and`, **`or`**	—	—	—	—	—	$2..*$
`not`	—	—	—	—	—	•

Table 6.1: Occurrence of Modifiers ($*$ indicates that the modifier has a scope of one to many terms, • exactly one term; — indicates that the modifier may not occur in that context)

Chapter 7

Programming in Xcerpt: Programs, Goals, and Rules

7.1 Xcerpt Programs

§1 Program

An Xcerpt program consists of at least one *goal* and some (possibly zero) *construct-query rules*.

Figure 7.1 shows an UML model for Xcerpt programs. For convenience, some of the input data of a program may be specified as part of the program using *data blocks* (similar to facts from logic programming, i.e., rules with an always successful query part.

§2 Goal

An Xcerpt goal specifies output of an Xcerpt program including an optional specification of where the output is to be stored.

Implementations must define default behavior for goals without output specification. More refined specifications for (XML and other) document properties as part of output specifications are under investigation, cf. Issue 24.

A program may contain multiple goals, allowing result to be stored in different files or at different Web locations. Notice, that the order of goals is currently undefined, cf. Issue 22, thus multiple goals with the same output target should be avoided for the time being.

§3 Rule

An Xcerpt *construct-query rule* (short: *rule*) relates a construct term to a top-level query term.

Rules can be seen as "views" specifying how to obtain documents shaped in the form of the construct term by evaluating the query against Web resources (e.g. an XML document or a database).

Recursive chaining of rules is possible (but note certain restrictions on recursion, cf. Section 7.2). In contrast to the inherent structural recursion used e.g. in XSLT, which is essentially limited to the tree structure of the input document, recursion in Xcerpt is always explicit and free in the sense that any kind of recursion can be implemented. Applications of recursion on

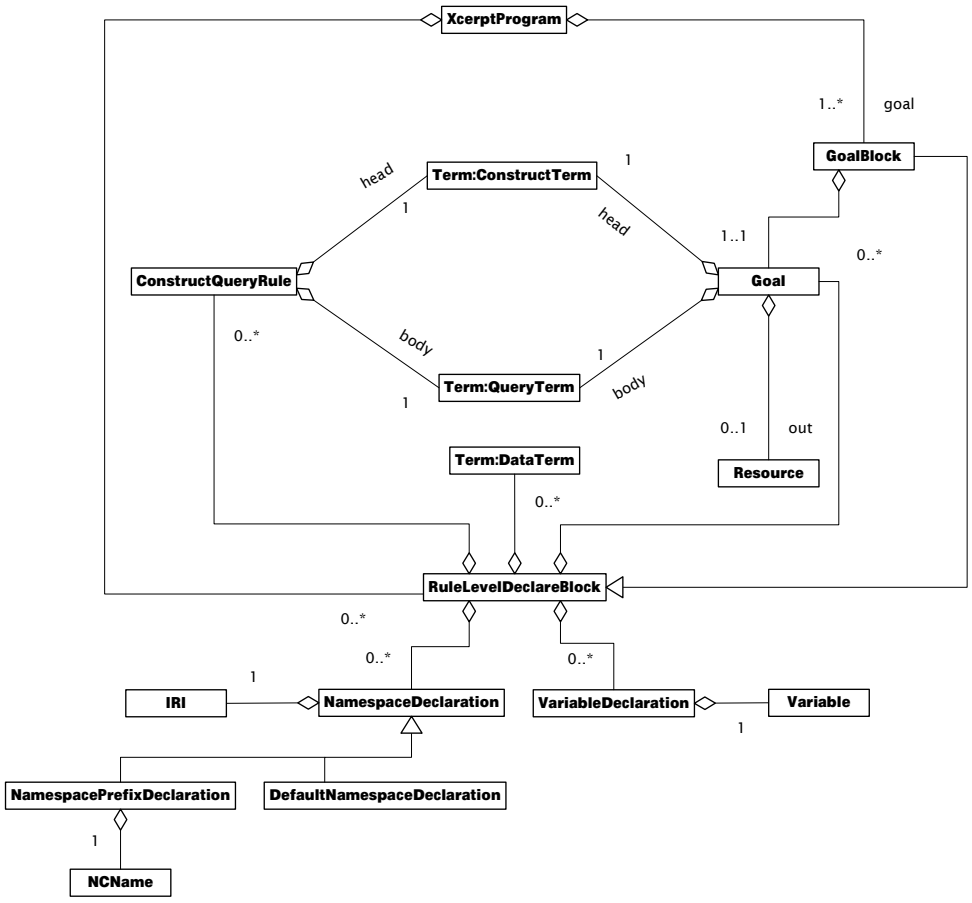


Figure 7.1: UML Model for Xcerpt Programs

the Web are manifold:

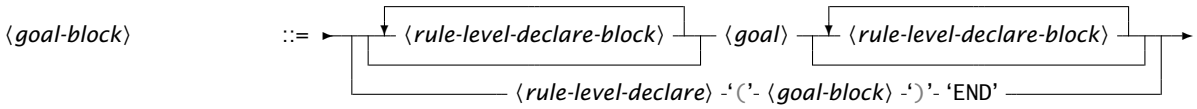
- structural recursion over the input tree (like in XSLT) is necessary to perform transformations that preserve the overall document structure and change only certain things in arbitrary documents (e.g. replacing all `em` elements in HTML documents by `strong` elements).
- recursion over the conceptual structure of the input data (e.g. over a sequence of elements) is used to iteratively compute data (e.g. create a hierarchical representation from flat structures with references).
- recursion over references to external resources (hyperlinks) is desirable in applications like Web crawlers that recursively visit Web pages.

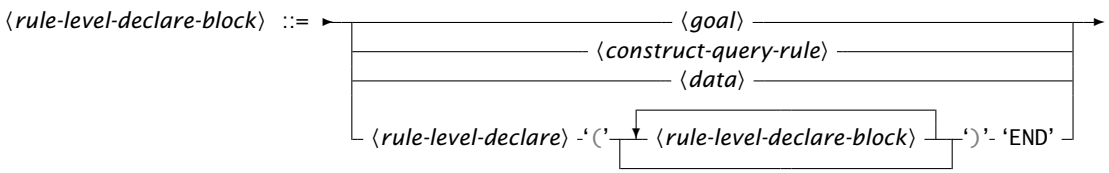
In addition to the syntactic constraints discussed so far, semantic constraints are imposed on the variables in rules, cf. Section 7.2.

7.1.1 Textual Term Syntax

The textual syntax for rules deviates from the term syntax by using uppercase keywords for block structures. This provides an easier visual distinction of rule- and term-level constructs.

⟨*program*⟩ ::= ‘PROGRAM’ -‘(’- ⟨*goal-block*⟩ -‘)’- ‘END’



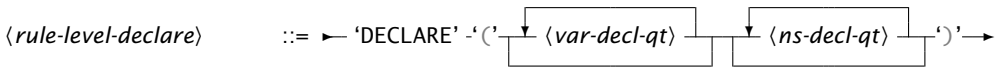


⟨*goal*⟩ ::= ‘GOAL’ -‘(’- ⟨*out-resource*⟩ -‘)’- ‘FROM’ -‘(’- ⟨*query-term*⟩ -‘)’- ‘END’

⟨*rule*⟩ ::= ‘CONSTRUCT’ -‘(’- ⟨*construct-term*⟩ -‘)’- ‘FROM’ -‘(’- ⟨*query-term*⟩ -‘)’- ‘END’

⟨*out-resource*⟩ ::= construct-term | ‘out’ -(- ⟨*iri*⟩ -|- ⟨*literal-var*⟩ -)‘(’construct-term‘)’

⟨*data*⟩ ::= ‘DATA’ -‘(’- ⟨*data-term*⟩ -‘)’- ‘END’



7.1.2 XML-style Term Syntax

The XML-style term syntax follows the same grammar as the non-XML term syntax. The full grammar is given in Section B.5.

7.1.3 Pure XML Syntax

In the pure XML syntax programs, goals, rules, and data blocks are very much straight forward and make extensive use of Relax NG's parameterizable grammars for declaration blocks. Figure 7.2 gives a graphical representation of the following Relax NG grammar.

```
## An Xcerpt program is a set of (one or more) goals, as well as (any number of) rules and inline data terms (like facts in Prolog). Rules and data terms may be surrounded by declaration blocks.
program = element program { goal-block }
goal-block =
  rule-level-block*
  | goal
  | rule-level-block*
  | grammar {
      include "declare-block.rnc" {
        content = parent goal-block
      }
    }

## Rule-level blocks form the basic block structure of an Xcerpt programs: goals, rules, and inline data terms form the basic block structures. They can be included into declaration blocks that define the scope of variable and namespace declarations.
rule-level-block =
  goal
  | rule
  | data
  |
    ## A declaration block on rule level allows both variable and namespace declarations.
    grammar {
      include "declare-block.rnc" {
        content = parent rule-level-block*
      }
    }

## A rule specifies how from data matched by the query term new data is constructed according to a construct term.
rule =
  element rule {
    element construct { construct-term },
    element from { query-term }
  }

## A goal is a rule, where the resulting data is written to a specified resource. Hence, goals are not chained.
goal =
  element goal {
    element out {
      (variable-ct
       | attribute value {
          text
          >> a:documentation [
               "This should in-fact be a IRI as by RFC 3987. Since XML Schema datatypes only provides the anyURI datatype for URIs conforming the older RFC 2396, arbitrary text is allowed."
             ]
        }),
      element construct { construct-term }
    },
    element from { query-term }
  }
```

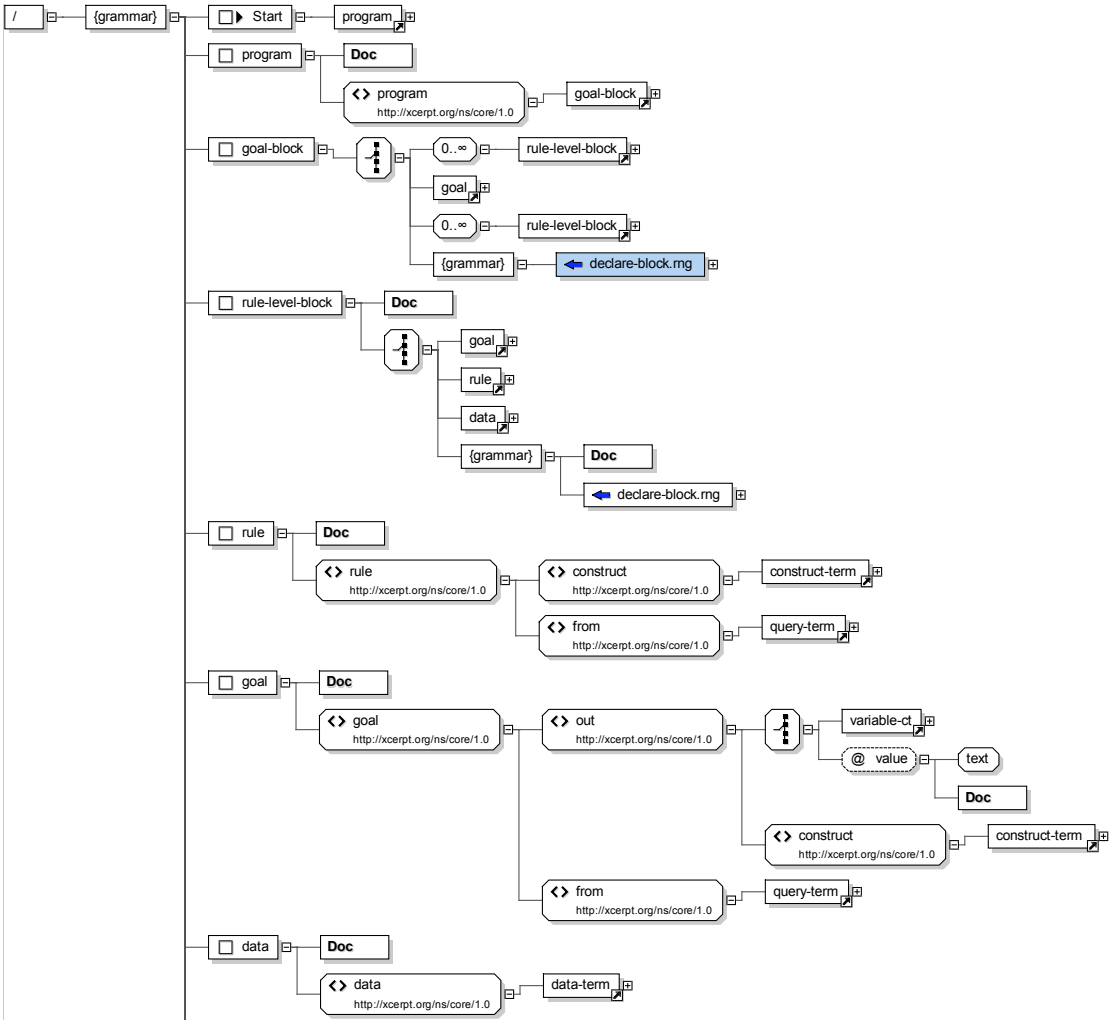


Figure 7.2: Relax NG Grammar for Xcerpt Programs

```
49 ## An inline data term allows the specification of data terms inside Xcerpt programs similar to facts in Prolog.
   data = element data { data-term }
```

7.2 Semantic Restrictions on Xcerpt Programs

Xcerpt imposes two major semantic restrictions on valid programs: range restrictedness and negation/grouping stratification. Where range restrictedness is a "local" property of a rule, negation and grouping stratification is a (global) property of an entire program.

Intuitively, range restrictedness ensures that all variables used in the construction part of a rule are properly bound in the query part. Negation and grouping stratification, on the other hand, disallow programs with a recursion over negated queries or grouping constructs, thus allowing an easier declarative semantics for Xcerpt programs.

optional or except variable only inside optional in head ERR optional should contain at least one variable WAR

or where the arity of the variable sets of the parts is not the same

7.2.1 Range Restrictedness

Intuitively, range restrictedness means that a variable occurring in a rule head also must occur at least once in the rule body. This requirement simplifies the definition of the formal semantics of Xcerpt, as it allows to assume that all query terms are unified with data terms instead of construct terms (i.e., variable free and collection free terms). Without this restriction, it is necessary to consider undefined or infinite sets of variable bindings, which would be a difficult obstacle for a forward chaining evaluation. Besides this formal reason, range restricted programs are also usually more intuitive, as they disallow variables in the head that are not justified somewhere in the body.

The following sections give a formal, syntactic criterion for range restrictedness, which considers negated queries and optional subterms as well as disjunctions in rule bodies.

7.2.1.1 Polarity of Subterms

So as to determine whether a rule is range restricted, variable occurrences in query and construct terms are associated with the polarities *positive* (+) or *negative* (−), and the attributes *optional* (?) or *not optional* (!) for such variables that are contained within an optional subtree and thus are not bound in all valid matchings. Intuitively, a *negative* variable occurrence is a *defining* occurrence, whereas a *positive* variable occurrence is a *consuming* occurrence. Since most terms are considered to be not optional, the attribute ! is omitted in most examples.

The polarity of variable occurrences in a term can be determined by recursively attributing all subterms of a term.

Polarity of Subterms

1. Let t be a query term with polarity p and optionality o.
   - if t is of the form `without` t', then t' is of polarity + (regardless of p) and optionality o

- if t is of the form **optional** t', then t' is of polarity p and optionality ?.
- if t is of the form $\langle modifier \rangle$ t', where $\langle modifier \rangle$ is not an occurrence modifier (i.e., neither **optional** nor **without**), then then t' is of polarity p and optionality o (*unchanged*).
- if t is of the form $var\ X \rightsquigarrow t'$ then t' is of polarity p and optionality o (*unchanged*).
- if t is of one of the forms $l\{\{t'_1, \ldots, t'_n\}\}$, $l\{t'_1, \ldots, t'_n\}$, $l[[t'_1, \ldots, t'_n]]$ or $l[t'_1, \ldots, t'_n]$ ($n \geq 0$), then $t'_1, \ldots, t'_n$ are of polarity p and optionality o (*unchanged*).
- if t is of the form t' **where** c with c a condition formula, then t' is of polarity p and optionality o, c is of polarity + and optionality o (*conditions are always consuming*).
- if c is a condition formula of the form **optionl** c' with c' a condition formula, then c' is of polarity p and optionality ?.

The rules for attribute terms are analogously, i.e., **optional** forces the optionality to ?, **without** the polarity to +, all other modifiers leave them unchanged.

2. Let t be a construct or data term with polarity p and optionality o.

   - if t is of the form **optional** t', then t' is of polarity p and optionality ?.
   - if t is of the form **optional** t' **with-default** t'', then t' is of polarity p and optionality ? and t'' is of polarity p and optionality o.
   - if t is of one of the forms $f\{t'_1, \ldots, t'_n\}$ or $f[t'_1, \ldots, t'_n]$ ($n \geq 0$), then $t'_1, \ldots, t'_n$ are of polarity p and optionality o.
   - if t is of the forms **all** t' or **some** t', then t' is of polarity p and optionality o.
   - if t is of the form $op(t'_1, \ldots, t'_n)$, with op a function or aggregation identifier, then $t'_1, \ldots, t'_n$ are of polarity p and optionality o.

The root of a query term is usually of negative polarity (and thus define variable bindings), as query terms usually occur in rule bodies. The root of a construct or data term is usually of positive polarity.

In a rule, the construct term in the head always has positive polarity and the query part has negative polarity and both are, by default, not optional. If negation constructs occur, the polarity changes accordingly. Furthermore, if parts of a query are negated by **not**, the polarity of these parts is again positive:

Polarity in Rules

1. If $R = t^c \leftarrow Q$ is a rule or goal with t^c a construct term and Q a query part, then the polarity of t^c is + and the polarity of Q is −.

2. Let Q be a query part with polarity p.

   - if Q is of the form $not\ Q'$, then Q' is of polarity + (regardless of p)
   - if Q is of the forms $and(Q_1, \ldots, Q_n)$ or $or(Q_1, \ldots, Q_n)$, then Q_1, ..., Q_n are of polarity p
   - if Q is a document specification with content C, then all variables occuring in the document specification are of polarity + and C is of polarity p.

- if Q is of the form t (a query term), then t is of polarity p.

Note that the polarity of negated subterms and queries is *always* positive, regardless of the level of nesting. The rationale behind this is that, since negation in Xcerpt is *negation as failure* and not the negation of classic logic, additional negations do not completely revert previous negations. Variable occurrences that are in the scope of at least one negation construct are always consuming occurrences, since negation as failure requires to perform auxiliary computations.

Returning to the definition of range restrictedness, it requires that in a rule, for each consuming occurrence of a variable, there exists at least one defining occurrence. Furthermore, a variable for which all defining occurrences are optional also needs to be optional on all consuming occurrences. This restriction is straightforward to understand, as it just requires that *"each variable in the head or in a negated query needs to be bound elsewhere"*.

This intuitive definition of range restrictedness is complicated by the possibility of disjunctions in the rule body, in which case a variable occurring positively in the rule head needs to occur negatively in *each* disjunct. Since disjunctions can also be nested, it is useful to define a *disjunctive rule normal form*, cf. [38].

Given a rule in such a disjunctive rule normal form, range restrictedness requires that each variable that occurs positively in one of the disjuncts occurs also negatively in the same disjuncts. Range restrictedness is formalised by the following definition:

Range Restrictedness Let R be a rule or goal and let $R' = t^c \leftarrow Q_1 \vee \ldots \vee Q_n$ ($n \geq 0$) be the disjunctive rule normal form of R. R is said to be *range restricted*, iff

1. for each disjunct Q_i ($1 \leq i \leq n$) holds that each variable occurring with positive polarity in either t^c or Q_i also occurs at least once with negative polarity in Q_i.

2. each variable that occurs in at least one of the Q_i ($1 \leq i \leq n$) attributed as as *optional* without another non-optional, negative occurrence in Q_i[1] or that occurs in Q_i, but not in Q_j for some $1 \leq i, j \leq n$ is attributed as optional in all positive occurrences in Q_i (including condition formulas and document specifications) and t^c.

A program P is called *range restricted*, if all rules $R \in P$ are range restricted.

7.2.2 Negation and Grouping Stratification

Stratification is a technique first proposed by Apt, Blair, and Walker [2] to define a class of logic programs where non-monotonic features like Xcerpt's grouping constructs or negation can be defined in a declarative manner. The principal idea of stratification is to disallow programs with a recursion over negated queries or grouping constructs and thereby precluding undesirable programs. While this requirement is very strict, its advantages are that it is straightforward to understand and can be verified by purely syntactical means without considering terms that are not part of the program. Several refinements over stratification have been proposed, e.g. *local stratification* [36] that allow certain kinds of recursion, but these usually require more "knowledge" of the program or the queried resources.

[1] Notice, that the optional occurrence must be of negative polarity in this case, as the query term would otherwise contradict rule (1).

Xcerpt programs in this thesis are considered to be stratifiable[2]. Furthermore, the notion of stratification is not only used for proper treatment of negation, it also extends to rules with grouping constructs, because a recursion over grouping constructs usually defines undesirable behaviour. A detailed discussion of stratification in Xcerpt can be found in [38, 37].

Here, we only give the final definition of a fully stratified Xcerpt program:

Full Stratification of an Xcerpt Program Let $P = P_1 \uplus \ldots \uplus P_n$ denote a *partitioning* of a set P into disjoint sets P_i. Given a program P consisting of rules/goals $\{R_1, \ldots, R_m\}$ ($m \geq 1$).

1. Let $R = t^c \leftarrow Q$ and $R' = t^{c'} \leftarrow Q'$ be rules.

   - R *depends* on R' if there exists a (negated or non-negated) query term t^q in Q such that t^q simulation unifies in $t^{c'}$
   - R *depends positively* on R' if there exists a non-negated query term t^q in Q such that t^q simulation unifies in $t^{c'}$
   - R *depends negatively* on R' if there exists a negated query term $not\ t^q$ in Q such that t^q simulation unifies in $t^{c'}$

2. P is called *fully stratifiable* (or simply *stratifiable*), if there exists a partitioning ($n \geq 1$)

   $$P = P_1 \uplus \ldots \uplus P_n$$

   of P such that for every stratum P_i ($1 \leq i \leq n$) and for every rule $R \in P_i$ holds:

   - if R depends negatively on a rule R', or the head of R contains grouping constructs and R depends positively or negatively on R', then $R' \in \bigcup_{j<i} P_j$, i.e. R' is in a strictly lower stratum than R
   - if the head of R contains no grouping constructs and R depends positively on a rule R' then $R' \in \bigcup_{j \leq i} P_j$, i.e. R' is in the same or in a lower stratum than R

The partition $P = P_1 \uplus \ldots \uplus P_n$ is called a *full stratification* of P, and the P_i are called *strata* of P.

[2] Rather than calling a program *stratified* as in the original definition, we call it *stratifiable* as it is not necessary to compute the stratification during (backward chaining) evaluation.

Part II

Language Extensions and Open Issues

Chapter 8

Open Issues: Language Constructs

The previous sections define the syntax for the language Xcerpt separated in (a) literal structures, (b) data, construct, and query terms, and (c) programs, goals, rules, and data blocks. Where appropriate, open issues are referenced in these sections. The remaining sections list those open issues and give a brief discussion. It is expected that these open issues are gradually resolved and their resolution integrated into the language Xcerpt.

This chapter is structured into issues related to general language constructs (Section 8.1), into issues on specific data representation formats such as XML or RDF (Section 9, into issues specific to one of the syntaxes for Xcerpt introduced above (Section 10), and finally into language extensions (Section 11).

8.1 General Issues

Issue 1. Substitution Sets vs. Substitution Multi-Sets
Currently, the Xcerpt semantics uses substitution sets, but the description of construct terms presented in Section 5 uses multi-sets. In the case of *sets*, duplicate substitutions are removed. Though this is useful in some queries, there are many cases where this is not necessary or even detrimental:

1. First, duplicate removal is a costly operation. Though (node identity-based) duplicates can be avoided at small additional cost for queries against trees (as shown for XPath evaluation), this is not true for general queries against graphs. Duplicate removal is cubic in the number of substitutions for the case of Xcerpt, as it is based on structural equivalence.

2. Second, for many types of queries duplicate removal is not wanted, e.g., in transformation and aggregation queries: It is currently not possible to express such simple and straightforward queries as "Count the number of `title` elements in a document" or "Replace all `a` labeled elements in the input tree with `b` labeled elements otherwise retaining the same structure" without resorting to complicated rules using **`position`**.

Solution Proposal(s): A more proper solution for handling duplicates seems to be in grouping expressions which can be "parameterized" by different requirements, e.g., duplicate elimi-

nation based on structural equality, duplicate elimination based on node identity, no duplicate elimination.

A more detailed proposal is needed.

Resolution: Though the use of substitution multi-sets and "parameterized" grouping expressions is desirable from a programmers point of view, the consequences for Xcerpt's semantics need further consideration.

Issue 2. Optional for Non-Term Variables

Occurrence modifiers (**`without`** and **`optional`**) are currently only allowed on entire term expressions. However, namespaces and term identifier are also non mandatory parts of a term, that might be optionally queried. It is currently not possible to query, e.g., for all nodes without term identifiers.

Solution Proposal(s): For namespaces this issue could be resolved by referring to the empty namespace (represented by the empty string). However, this introduces another "optionality" concept into the language.

Alternatively, **`optional`** and **`without`** could also be allowed on non-term expressions. Then a careful consideration where is needed (e.g,. document specifications).

This issue is related to Issue 3.

Resolution: Not yet reached.

8.1.1 Defaults and Default Modes

Issue 3. Absent attribute and children lists and other parts of a term specification

A number of parts in term specification are optional, i.e., may or may not occur or may or may not be empty. These include:

- the lists of attributes and children,
- the namespace and identifier of a term,
- various parts of a XML document specification, and
- comments and processing instructions.

Currently, attribute and children lists may not be omitted, even if they are empty. Namespaces and term identifiers may be omitted, but where no namespace in a query is equivalent to an empty namespace, missing term identifiers do not affect matching. No special consideration is given to comments and processing instructions, i.e., e.g., a total query term matches no data terms that are matches up to comments.

Solution Proposal(s): A more "unified" policy for defaults should be enacted. In particular, it might be worth considering "default modes" that the programmer can control. Similar to (rule-level) declaration blocks, a programmer could specify that in the scope of the mode declaration, e.g., omitted attribute lists are to be considered as `()` or as `(( ))`.

A more detailed proposal is needed.

Resolution: Not yet reached.

8.2 Construct Specific Issues

8.2.1 Conditional Construction and `optional` Construct Terms

Issue 4. Conditional Construction

Optional construct terms allow one form of conditional construction, viz.where the condition is that bindings for all optional variables of the optional construct term exist. The default value plays are role similar to the `else` clause in `if ... then ... else` expressions.

However, this is a very limited form of conditional construction that also exhibits some anomalies:

1. **`optional`** is the only modifier allowed in query and construct terms, though with a slightly different meaning. E.g., an optional ground term in a query term may or may not occur in the data, but an optional ground term in a construct term always occurs in the result.

2. It allows only implicit specification of optional variables (in contrast to grouping), i.e., it is not possible to make the optional part dependent on a variable not occurring in the result. This makes it difficult to express the following query:

```
for $x in doc("books.xml")/bookstore/book
return if ($x/@category="CHILDREN")
      then <child>{data($x/title)}</child>
      else <adult>{data($x/title)}</adult>
```

3. When the choice depends on the value of a optional variable this can be expressed in a condition box, but only in the case of exactly two choices, not if there are more alternatives.

```
CONSTRUCT
  result [
    optional ( child(opt from-year=var Year)[ var Title ] ) with-default ( adult [ var
        Title ] )
  ]
FROM
  bookstore [[
    book (from-year=opt var Year)[[ var Title →title [[]]]
  ]] where opt (var Year > 25)
END
```

Solution Proposal(s): A proposed solution is the introduction of a `if ... then ... else` or `case` expression into construct terms, that allows for conditions as in **`where`** clauses. The requirement that matching is never affected by the construct part of a rule should be upheld.

A more detailed proposal is needed.

Resolution: Not yet reached.

8.2.2 Query Formulas as Subterms

Issue 5. Query Formulas as Subterms

Currently, query formulas (expressions using **`and`**, **`or`**, and **`not`**) are only allowed at the top-level of query terms. However, programs become both more compact and easier to read, as well as easier to efficiently evaluate, if formulas are allowed at sub-term level.

However, this introduces a number of questions:

- What becomes then the different between **not** and **without**? If **not** is allowed at sub-term level, **without** seems to be superfluous.
- What is then the difference between $t[[a, b]]$ and $t[[and(a, b)]]$? Only that the injectivity constraint between a and b is lifted?

Solution Proposal(s): *A detailed proposal is needed.*
Resolution: Not yet reached.

8.2.2.1 `withouts` as Direct Siblings

Issue 6. Without Siblings
If several direct siblings in an ordered term or arbitrary siblings in an unordered term are modified by **without**, the semantics of the expression becomes unclear: E.g., matches `f[[a, without b, without c, d]]` with `f[a,c,b,d]` or not? In other words how are order and injectivity constraints enforced between **without**-modified terms.

Solution Proposal(s): A possible solution for this case is the use of **or** to clarify the semantics of the expression. However this hinges on a positive solution of Issue 5.

Resolution: Not yet reached.

8.2.3 Functions and Libraries: Built-In and User-defined

Issue 7. Relational vs. functional Operators
Xcerpt does not have an extensive function library as of now. Introducing such a library requires great care, in particular when considering not only functional operators (such as compare, concat, or arithmetic operators), but also relational (sequence-valued) operators (such as tokenizers).

Solution Proposal(s): The latter are more similar to predicates (rules) where certain parameters are consuming, others are defining. The addition of rules where parameters are specifically marked as consuming only would solve these cases. Functional operators could also be handled this way, but differ in where they may be used. Functional operators are also useful in construct terms and in condition expressions.

Considering the sort of functions and operators to support, the XQuery and XPath function and operator library [30] is certainly a good starting point.

A more detailed proposal is needed.

Resolution: Not yet reached.

Issue 8. Expressions in Conditions
Optional modifiers are currently only allowed on variables in conditions, but should actually range over entire expressions.

Resolution: Not yet reached.

8.2.4 Combining and Comparing Modifiers

Issue 9. Removing Useless Modifier Combinations
Some combinations of modifiers in query terms are useless:

1. Any combination of **except** and **without** on the same term, e.g.: *var* X →a [except **without** b [ ]] *var* X →a [**without** except b [ ]]. Both examples are equivalent to *var* X →a [**without** b [ ]] as **except** does not affect the occurrence of the in-scope sub-term.

2. **optional** (anywhere) in the scope of **without**.

3. **except** outside variable restrictions

4. **except** within **except** without another variable restriction in-between.

Solution Proposal(s): Some of these could become warnings, since they are not strictly wrong but merely useless.

Resolution: Not yet reached.

Issue 10. Rewriting Optional

A canonical rewriting of optional should be introduced into the definition. The naive rewriting *var* X →a[[**opt** *var* Y →b]] to **or**(*var* X→a[[**without** b]], *var* X →a [[*var* Y →b]]) can lead to adjacent **without**, cf. Issue 8.2.2.1.

Resolution: Not yet reached.

Issue 11. Rewriting Without

A canonical rewriting of **without** should be introduced into the definition to clarify its semantics. Again this depends on the resolution of Issue 8.2.2.1.

Resolution: Not yet reached.

8.2.5 Variables

Issue 12. Variable Restrictions for Identifiers and Literal Content

Following [37], the text currently allows variable restrictions only on structured terms and attribute terms. This disallows in particular variable restrictions on label variables. The latter ones are useful just as variable restrictions on literal content, if regular expressions are used to restrict the set of labels or content nodes (or namespaces or any other place where an identifier may occur).

Solution Proposal(s): Note, that they are not strictly necessary due to variables in regular expressions, i.e., instead of writing *var* X →/⟨*reg-exp*⟩/ one may use /(*var* X →⟨*reg-exp*⟩)/.

Resolution: Not yet reached.

Issue 13. Separation of literal and term variables

The current version of the Xcerpt grammar does not strictly enforce a separation between literal and term variables. This is left to an eventual type system. However, it is possible to enforce this in the grammar (with the exception of literal content) and would be useful in absence of a full type system.

Resolution: Not yet reached.

8.2.6 Varia

Issue 14. Explicit Variable Specifications for Except

Except is defined to affect the bindings of all variables in restrictions for which it occurs. In the following query term bindings for both X and Y are affected (i.e., the c sub-term is excluded from bindings for both):

```
root [[ var X →a [[ var Y →b [[ except c [[ ]] ]] ]] ]]
```

Against the data term `root[ a[ b[c] ] ]` this results in one binding for `X`, viz. `a [ b [ ] ]` and one binding for `Y`, viz. `b [ ]`.

This is problematic as it makes impossible nested variable restrictions where one excludes some sub-term and another does not. In the example, it is not possible to affect the bindings to `Y` without affecting the bindings for `X`.

Solution Proposal(s): Introducing explicit "variable specifications" for **except**, i.e., an explicit list of variables in whose bindings the sub-terms in the scope of the **except** modifier are removed. Bindings of all other variables remain unaffected and must contain all sub-terms in the scope of the **except** (i.e., for them the **except** is simply ignored).

Resolution: Not yet reached.

8.3 Querying the Type of Data, Typed Accessors

There are a number of issues related to an upcoming Xcerpt type system.

Issue 15. Typed Accessors and Coercion

For terms, it should be possible to access (a) the actual structure, (b) the typed value of the term, if it has any, and (c) the string value of the term (defined, e.g., as in XPath).

Resolution: Not yet reached.

Issue 16. Typing Data Terms

It should be possible to explicitly type data terms, e.g., to distinguish plain strings from strings representing date.

Resolution: Not yet reached.

Issue 17. Querying Typed Data

It should be possible to query data based on its type. This is particularly helpful, if the type is complex, as it avoids the need for complex (and difficult to evaluate) patterns in these cases.

Resolution: Not yet reached.

Issue 18. List-valued Attributes and Content

As discussed in Issue 8.3, some conventions for representing lists or sets of atomic values as, e.g., whitespace-separated strings have surfaced, e.g., attributes of type IDREFS in XML [8] or XML Schema [22] list types (cf. Section of [5]).

Both typing of variables and terms bound or containing such data and extraction of atomic values from such data should be supported eventually.

Resolution: Not yet reached.

8.4 Node Identity and Term Identifiers in Xcerpt

Issue 19. Node identity

Currently, the concept of node identity is not consistently supported in Xcerpt.

When doing structural recursion over an input document, it is often necessary to differentiate between two subterms that share the exact same structure but appear at different positions

in the input document. For instance, without considering the identity of subterms, the **all** construct would coalesce structurally and value equivalent bindings and join operations using **and** could lead to unexpected results.

Solution Proposal(s): A new construct **identity** is proposed, allowing access to identifiers uniquely identifying each subterm. A query term of the form **identity** *var* X a{{ }} matches the same data terms as the query a{{ }}, but also binds an identifier unique to each data term to the variable *X*.

Identity specification is admissible in any kinds of query term specifications, e.g in ordered and unordered as well as total and partial query terms as well as in combination with constructs like **descendant**. Note, however, that it is not possible to specify a constant instead of a variable, as the structure and form of the identifiers is implementation-dependent and the identifiers are only suited for comparison between multiple bindings.

The identity of a data term is a property of the term that does not change for the entire lifetime of the data term, similar to other properties like sorted/unsorted or total/incomplete. The uniqueness of the identifiers defined as follows:

- an identifier for a data term originating in an external resource is always the same, even if the same resource is retrieved in multiple rules,
- the identifier does not change when a data term is matched by and passed through a rule,
- when multiple structurally and value equivalent data terms are merged by grouping, the resulting data term and its subterms have the identity of any one of the merged data terms and its subterms,
- the identifier for a single data term only needs to be unique for a single query of a single instance of an Xcerpt implementation.

This proposal shows possible problems: should the identity of constructed terms really be the same as the one of original ones? What about duplication in the construction? Then identity is not even "locally" unique any more? What about grouping?

A better alternative might be that constructed terms have always new identity. This however causes potentially problems for optimizations that remove rules.

Also the use of **identity** is painful in queries where the only objective is avoiding duplicate elimination in grouping.

A more detailed proposal is needed.

Resolution: Not yet reached.

8.4.1 Scope of Term Identifiers and Cross-Document References

Issue 20. Cross-Document References

Term identifiers are defined to be unique in the context of the data unit or document they are contained in. In many contexts, globally unique identifiers can be very useful, e.g., when considering RDF data. Also if multiple documents are accessed together, a natural extension of Xcerpt's current reference handling could be some mechanism for transparent reference resolution for out-document links. Or maybe two concepts are needed: globally unique term identifiers such as URIs in RDF and in-document IDs as in XML, where the latter are very convenient when writing terms (you do not have to care about the rest of the world) and can be automatically "lifted" to globally unique identifiers.

Resolution: Not yet reached.

8.4.2 Collapsing Text Nodes

Issue 21. Consecutive Text Nodes Collapse

Currently, consecutive text nodes are (silently) collapsed at construction. This is in accordance to the XML data model. Is this the desired behavior? Should we allow consecutive text nodes and provide an explicit concatenation?

Resolution: Not yet reached.

8.4.3 Goal Order

Issue 22.

Are goals evaluated in a particular order? What if one goal modifies data used in another? What if two goals write to the same output?

Resolution: Not yet reached.

8.4.4 Document Specifications

Issue 23. Document Specifications

Currently, document specifications are specifically treated in the grammar, which makes the grammar rather bloated. It could be better to consider them as a canonical "transformation" to terms. However, this would make the enforcement of constraints such as all variables in document specifications are consuming only more difficult.

Resolution: Not yet reached.

Chapter 9

Open Issues: Specific to Data Representation Format

9.1 Serializing to XML and from XML

Issue 24. Serializing to XML
Similar to the XML document specification in query terms, there should be a document serialization specification in goals describing how to serialize the resulting term. It might be worth considering the adoption of the XQuery/XSLT Serialization Recommendation [29]. Additionally, a canonical representation of graph structures, non XML labels, contents of comments or processing-instructions that are not XML conform, unordered elements, adjacent text nodes (if allowed, cf. Issue 21) must be defined.

Resolution: Not yet reached.

Issue 25. Accessible Encoding of XML Documents
Currently, the XML document specification does not allow access to the encoding of the file in accordance to the XML Information Set recommendation.

Resolution: Not yet reached.

Issue 26. XML Base
Xcerpt does not support the XML Base specification [31], i.e., no base URL at elements as in the XML Information Set.

Consider the extraction of some element at a non-root level. Now relative references are not any longer resolvable as the connection to the root-level xml:base attribute is lost.

Resolution: Not yet reached.

Issue 27. In-scope Namespaces
There are a number of issues related to namespaces in XML in general and Xcerpt in particular:

1. exactly what strings are permitted as namespace URIs? Different W3C specifications differ on this point.

2. Are namespace declarations information-bearing? For example, are the two documents below equivalent: (1) `<a xmlns:x="x"><b/></a>` (2) `<a xmlns:x="x"><b xmlns:x="x"/></a>`

3. Are in-scope namespaces that are not referenced information-bearing? For example, are either of the above documents equivalent to:
   (3) `<a><b/></a>`?
   The main problem here, of course, is "QNames in content": the use of namespace-sensitive element and attribute values. But there are also applications that use the mere presence of a namespace declaration as a flag or marker.

4. Are prefixes information-bearing? That is, is document (1) equivalent to: (4) `<a xmlns:y="x"><b/></a>`
   Again, the main problem is "QNames in content".

5. In the light of the above, how should namespaces be handled by applications that allow a document to be modified? For example, if an element is deep-copied from one place to another, should it take all its in-scope namespace declarations with it?

Indeed the Canonical XML recommendation *dropped even namespace rewriting* for precisely these reasons: "The C14N-20000119 Canonical XML draft described a method for rewriting namespace prefixes such that two documents having logically equivalent namespace declarations would also have identical namespace prefixes. The goal was to eliminate dependence on the particular namespace prefixes in a document when testing for logical equivalence. However, there now exist a number of contexts in which namespace prefixes can impart information value in an XML document. For example, an XPath expression in an attribute value or element content can reference a namespace prefix. Thus, rewriting the namespace prefixes would damage such a document by changing its meaning (and it cannot be logically equivalent if its meaning has changed)."

Resolution: Not yet reached.

9.2 Accessing RDF Documents

Issue 28. RDF Document Specification and Serialization

Similar to the case for XML, access to RDF needs a document specification and serialization facility in Xcerpt. This needs to consider among other things (a) the serialization format of RDF and (b) the RDF version.

Resolution: Not yet reached.

Chapter 10

Open Issues: Specific to Concrete Syntax

10.1 Non-XML Term Syntax

Issue 29. Nested Comments in Non-XML and XML-style Term Syntax
Currently, neither syntax allows nested comments. Though nested comments are commonly shunned in programming languages, there are some recent languages (REXX, Haskell, XQuery) that use nested comments. For a comparison of comments in programming languages cf. `http://www.gavilan.edu/csis/languages/comments.html`.

Resolution: Not yet reached.

Issue 30. IRIs in Recommended IETF Angle Bracket Notation
Currently, IRIs are not syntactically separated from strings. The IETF recommends URIs to be denoted in angle brackets, a notation used in many RDF formats. This clashes however with angle brackets in XML.

Resolution: Not yet reached.

10.1.1 Style Guide

Issue 31. Style Guide for Indenting, Naming, etc.
A style guide covering comments, nesting, indenting, use of brackets and commata, rules, document specifications, etc. should be specified. A similar undertaking is underway for XQuery.

Resolution: Not yet reached.

Issue 32. Comparison Function in Grouping Terms
Replacing the separate specifications for equivalence and order relation with a single three-valued comparison function (in the style of Java's `compareTo` function) makes the specification of the grouping term more compact, but requires that the used comparison function always deals with both equivalence and order.

Resolution: Not yet reached.

10.2 XML-style Term Syntax

Issue 33. Quoting in XML-style Term Syntax

Currently, strings are quoted in the XML-style term syntax. However, there is a strong motivation to make XML documents cut-and-pastable, thus requiring the adoption of the XML character encoding rules into the XML-style Term syntax. In this case, the issue of entities must be considered. Also, keywords must be quoted in this case, possibly using character sequences illegal in XML.

Resolution: Not yet reached.

10.3 Pure XML Syntax

No issues with the pure XML syntax have been identified so far.

Chapter 11

Open Issues: Language Extensions

11.1 RDF Querying in Xcerpt

A better support for RDF querying in Xcerpt is desirable. This entails in particular three issues:

Issue 34. b-Nodes: Variables in Data Terms
RDF allows existentially quantified variables in data terms (in the form of b-nodes). This is currently not considered by Xcerpt though matching against construct terms is indeed part of the Xcerpt evaluation (at least for backward chaining) and should provide the needed methods.
Resolution: Not yet reached.

Issue 35. Partial Data Terms
Also, RDF data terms are always partial, i.e., there may be more information at other locations on the Web. This would be particularly useful if one considers identifiers that are unique not just within a document but in an entire collection.
Resolution: Not yet reached.

Issue 36. Proper Triple Syntax
So far, RDF is mapped into Xcerpt terms both internally and on the level of the syntax. It might be easier for programmers to provide a proper triple syntax as syntactic sugar.
Resolution: Not yet reached.

11.2 Modular Xcerpt

11.2.1 Modules and Components in Xcerpt

A more modular version of Xcerpt is currently under development, where rules can form modules and libraries that can be linked together (both statically and dynamically).

11.2.2 Macros, Abbreviations, Code Reuse

Xcerpt also lacks a proper macro or abbreviation mechanism as well as facilities for code reuse.

11.2.3 Web Service Access

Finally, access to Web Services (including other Xcerpt instances) provides a system of dynamic distribution of query loads as well as access to rich service APIs such as Amazon's or Google's.

11.3 Visual and Verbal Syntax for Xcerpt

The visual syntax for Xcerpt (visXcerpt) should be adapted to the current state of the syntax documented in this document.

A verbalization of Xcerpt and XML is under consideration and could make queries far more accessible.

Acknowledgements.

This research has been funded by the European Commission and by the Swiss Federal Office for Education and Science within the 6th Framework Programme project REWERSE number 506779 (cf.`http://rewerse.net`).

Part III

Full Language Grammars

Appendix A

Grammar for Non-XML Term Syntax

A.1 Literal Structures

⟨*NCName*⟩ ::= ⟨*http://www.w3.org/TR/REC-xml-names/#NCName*⟩

⟨*IRI*⟩ ::= '"' ⟨*http://www.ietf.org/rfc/rfc3987.txt#IRI*⟩ '"'

⟨*String*⟩ ::= '"' ⟨*StringCharacter*⟩* '"'

⟨*StringCharacter*⟩ ::= ⟨*http://java.sun.com/docs/books/jls#StringCharacter*⟩ | ⟨*Line-feed*⟩
| ⟨*Carriage-return*⟩

⟨*Line-feed*⟩ ::= '\u000a'

⟨*Carriage-Return*⟩ ::= '\u000d'

⟨*Number*⟩ ::= ⟨*int*⟩

⟨*Int*⟩ ::= ⟨*http://www.w3.org/TR/REC-xml-names/#Digit*⟩*

⟨*Regexp*⟩ ::= '/' ⟨*{http://www.unix.org/version3/ieee_std.html#}extended_reg_exp*⟩
'/'

⟨*ERE_expression*⟩ ::= ⟨*http://www.unix.org/version3/ieee_std.html#one_char_or_coll_elem_ERE*⟩
| '^'
| '$'
| '(' ⟨*http://www.unix.org/version3/ieee_std.html#extended_reg_exp*⟩
')'
| '(' ⟨*variable*⟩ '->' ⟨*http://www.unix.org/version3/ieee_std.html#extended_reg_exp*⟩')'
| ⟨*http://www.unix.org/version3/ieee_std.html#ERE_expression*⟩
⟨*{http://www.unix.org/version3/ieee_std.html#}ERE_dupl_symbol*⟩

⟨*Whitespace*⟩ ::= (⟨*http://www.w3.org/TR/REC-xml/#S*⟩ | ⟨*End-of-line-comment*⟩ | ⟨*Block-comment*⟩)*

⟨*Comment-char*⟩ ::= ⟨*http://www.w3.org/TR/REC-xml/#NT-Char*⟩

⟨*End-of-line*⟩ ::= ⟨*Line-feed*⟩ | ⟨*Carriage-return*⟩ (⟨*Line-feed*⟩)?

⟨*End-of-line-comment*⟩ ::= '#' (⟨*Comment-char*⟩* - (⟨*Comment-char*⟩* ⟨*End-of-line*⟩ ⟨*Comment-char*⟩*) ⟨*End-of-line*⟩

⟨*Block-comment*⟩ ::= '/#' (⟨*Comment-char*⟩* - (⟨*Comment-char*⟩* ('/#' | '/#') ⟨*Comment-char*⟩*) '/#'

A.2 Data Terms

⟨*data-term*⟩ ::= ⟨*term-level-declare-dt*⟩ | ⟨*reference-dt*⟩ | ⟨*structured-dt*⟩ | ⟨*content-dt*⟩

⟨*reference-dt*⟩ ::= '^' ⟨*identifier-dt*⟩

⟨*term-level-declare-dt*⟩ ::= 'declare' ⟨*ns-declaration-dt*⟩ '(' (⟨*data-term*⟩ ','?)* ')'

⟨*identifier-dt*⟩ ::= ⟨*NCName*⟩ | ⟨*IRI*⟩ | ⟨*String*⟩

⟨*ns-declaration-dt*⟩ ::= (⟨*ns-prefix-declaration-dt*⟩ ','?)* (⟨*ns-default-declaration-dt*⟩ ','?)? (⟨*ns-prefix-declaration-dt*⟩ ','?)*

⟨*ns-prefix-declaration-dt*⟩ ::= 'ns-prefix' ⟨*identifier-dt*⟩ '=' ⟨*IRI*⟩

⟨*ns-default-declaration-dt*⟩ ::= 'ns-default' ⟨*IRI*⟩

⟨*content-dt*⟩ ::= ⟨*literal-content-dt*⟩ | ⟨*comment-dt*⟩ | ⟨*processing-instruction-dt*⟩

⟨*literal-content-dt*⟩ ::= ⟨*String*⟩

⟨*comment-dt*⟩ ::= 'xcerpt' ':' 'comment' '(' ')' '[' ⟨*literal-content-dt*⟩ ']'

⟨*processing-instruction-dt*⟩ ::= ('xcerpt' ':' 'processing-instruction' | 'xcerpt' ':' 'pi') '(' 'target-name' '=' ⟨*identifier-dt*⟩ ')' '[' ⟨*literal-content-dt*⟩ ']'

⟨*structured-dt*⟩ ::= ⟨*local-spec-dt*⟩ ⟨*children-list-dt*⟩

⟨*children-list-dt*⟩ ::= '[' (⟨*data-term*⟩ ','?) * ']'
| '{' (⟨*data-term*⟩ ','?)* '}'

⟨*local-spec-dt*⟩ ::= ⟨*term-identifier-dt*⟩? ⟨*ns-label-dt*⟩ ⟨*attr-term-list-dt*⟩

⟨*term-identifier-dt*⟩ ::= ⟨*identifier-dt*⟩ '@'

⟨*ns-label-dt*⟩ ::= (⟨*identifier-dt*⟩ ':')? ⟨*identifier-dt*⟩

⟨*attr-term-list-dt*⟩ ::= '(' (⟨*attr-term-dt*⟩ ','?)* ')'

⟨*attr-term-dt*⟩ ::= ⟨*base-attr-term-dt*⟩

⟨*base-attr-term-dt*⟩ ::= ⟨*ns-label-dt*⟩ '=' ⟨*literal-content-dt*⟩

⟨*top-level-data-term*⟩ ::= ⟨*top-term-level-declare-dt*⟩ | ⟨*structured-dt*⟩

⟨*top-term-level-declare-dt*⟩ ::= 'declare' ⟨*ns-declaration-dt*⟩ '(' ⟨*top-level-data-term*⟩ ')'

A.3 Construct Terms

⟨*construct-term*⟩ ::= ⟨*term-level-declare-ct*⟩ | ⟨*reference-ct*⟩ | ⟨*structured-ct*⟩ | ⟨*content-ct*⟩
| ⟨*term-variable-ct*⟩
| ⟨*modified-ct*⟩

⟨*reference-ct*⟩ ::= '^' ⟨*identifier-ct*⟩

⟨*term-level-declare-ct*⟩ ::= 'declare' ⟨*ns-declaration-ct*⟩ '(' (⟨*construct-term*⟩ ',')* ')'

⟨*identifier-ct*⟩ ::= ⟨*NCName*⟩ | ⟨*IRI*⟩ | ⟨*String*⟩ | ⟨*literal-variable-ct*⟩

⟨*ns-declaration-ct*⟩ ::= (⟨*ns-prefix-declaration-ct*⟩ ','?)* (⟨*ns-default-declaration-ct*⟩ ','?)?
(⟨*ns-prefix-declaration-ct*⟩ ','?)*

⟨*ns-prefix-declaration-ct*⟩ ::= 'ns-prefix' ⟨*identifier-ct*⟩ '=' (⟨*IRI*⟩ | ⟨*literal-variable-ct*⟩)

⟨*ns-default-declaration-dt*⟩ ::= 'ns-default' (⟨*IRI*⟩ | ⟨*literal-variable-ct*⟩)

⟨*content-ct*⟩ ::= ⟨*literal-content-ct*⟩ | ⟨*comment-ct*⟩ | ⟨*processing-instruction-ct*⟩

⟨*literal-content-ct*⟩ ::= ⟨*String*⟩ | ⟨*literal-variable-ct*⟩

⟨*comment-ct*⟩ ::= 'xcerpt' ':' 'comment' '(' ')' '[' ⟨*literal-content-ct*⟩ ']'

⟨*processing-instruction-ct*⟩ ::= ('xcerpt' ':' 'processing-instruction' | 'xcerpt' ':' 'pi')
'(' 'target-name' '=' ⟨*identifier-ct*⟩ ')' '[' ⟨*literal-content-ct*⟩ ']'

⟨*structured-ct*⟩ ::= ⟨*local-spec-ct*⟩ ⟨*children-list-ct*⟩

⟨*children-list-ct*⟩ ::= '[' (⟨*construct-term*⟩ ','?) * ']'
| '{' (⟨*construct-term*⟩ ','?)* '}'

⟨*local-spec-ct*⟩ ::= ⟨*term-identifier-ct*⟩? ⟨*ns-label-ct*⟩ ⟨*attr-term-list-ct*⟩

⟨*term-identifier-ct*⟩ ::= ⟨*identifier-ct*⟩ '@'

⟨*ns-label-ct*⟩ ::= (⟨*identifier-ct*⟩ ':')? ⟨*identifier-ct*⟩

⟨*attr-term-list-ct*⟩ ::= '(' (⟨*attr-term-ct*⟩ ','?)* ')'

⟨*attr-term-ct*⟩ ::= ⟨*base-attr-term-ct*⟩
| ⟨*term-variable-ct*⟩
| ⟨*modified-attr-term-ct*⟩

⟨*base-attr-term-ct*⟩ ::= ⟨*ns-label-ct*⟩ '=' ⟨*literal-content-ct*⟩

⟨*top-level-construct-term*⟩ ::= ⟨*top-term-level-declare-ct*⟩ | ⟨*structured-ct*⟩

⟨*top-term-level-declare-ct*⟩ ::= 'declare' ⟨*ns-declaration-ct*⟩ '(' ⟨*top-level-construct-term*⟩ ')'

⟨*variable-ct*⟩ ::= ⟨*term-variable-ct*⟩ | ⟨*literal-variable-ct*⟩

⟨*term-variable-ct*⟩ ::= ⟨*var-specification-ct*⟩

⟨*literal-variable-ct*⟩ ::= ⟨*var-specification-ct*⟩

⟨*var-specification-ct*⟩ ::= ('variable' | 'var')? ⟨*NCName*⟩

⟨*modified-ct*⟩ ::= ⟨*grouping-ct*⟩ | ⟨*optional-ct*⟩

⟨*modified-attr-term-ct*⟩ ::= ⟨*grouping-attr-term-ct*⟩ | ⟨*optional-attr-term-ct*⟩

⟨*grouping-ct*⟩ ::= ⟨*grouping-modifier*⟩ '(' (⟨*construct-term*⟩ ',')* ')' ⟨*groupby*⟩? ⟨*orderby*⟩?

⟨*grouping-attr-term-ct*⟩ ::= ⟨*grouping-modifier*⟩ '(' ⟨*attr-term-ct*⟩? ')' ⟨*groupby*⟩? ⟨*orderby*⟩?

⟨*grouping-modifier*⟩ ::= 'all'
| 'some' ⟨*number-ct*⟩
| 'first' ⟨*interval-ct*⟩

⟨*orderby*⟩ ::= 'order-by' '(' ((⟨*optional-variable*⟩ | ⟨*variable-ct*⟩) ',')* ')' ⟨*order-relation*⟩?

⟨*order-relation*⟩ ::= 'ascending' | 'descending' | ⟨*NCName*⟩

⟨*groupby*⟩ ::= 'group-by' '(' ((⟨*optional-variable*⟩ | ⟨*variable-ct*⟩) ',')* ')' ⟨*equivalence-relation*⟩?

⟨*equivalence-relation*⟩ ::= ⟨*NCName*⟩

⟨*optional-variable*⟩ ::= ⟨*optional-modifier*⟩ ⟨*variable-ct*⟩

⟨*interval-ct*⟩ ::= ⟨*number-ct*⟩ '-' ⟨*number-ct*⟩
| ⟨*number-ct*⟩ '-'
| '+'

⟨*number-ct*⟩ ::= ⟨*Int*⟩ | ⟨*literal-variable-ct*⟩

⟨*optional-ct*⟩ ::= ⟨*optional-modifier*⟩ '(' (⟨*construct-subterm*⟩ ',')* ')'
('with-default' '(' (⟨*construct-subterm*⟩ ',')* ')')?

⟨*optional-attr-term-ct*⟩ ::= ⟨*optional-modifier*⟩ '(' (⟨*attr-term-ct*⟩',')* ')'
('with-default' '(' (⟨*attr-term-ct*⟩ ',')* ')')?

⟨*optional-modifier*⟩ ::= 'optional' | 'opt'

A.4 Query Terms

⟨*query-term*⟩ ::= ⟨*modified-qt*⟩
| ⟨*term-level-declare-qt*⟩

⟨*term-level-declare-qt*⟩ ::= 'declare' ⟨*ns-decl-qt*⟩ ⟨*variable-decl-qt*⟩
'('(⟨*query-term*⟩ ',')* ')'

⟨*modified-qt*⟩ ::= ⟨*variable-term-qt*⟩
| ⟨*location-modified-qt*⟩
| ⟨*occurrence-modified-qt*⟩
| ⟨*selection-modified-qt*⟩

⟨*base-term-qt*⟩ ::= ⟨*reference-qt*⟩ | ⟨*content-qt*⟩ | ⟨*structured-qt*⟩

⟨*reference-qt*⟩ ::= '^' ⟨*identifier-qt*⟩

⟨*identifier-qt*⟩ ::= ⟨*NCName*⟩ | ⟨*IRI*⟩ | ⟨*String*⟩
| ⟨*literal-variable-qt*⟩
| ⟨*Regexp*⟩

⟨*ns-decl-qt*⟩ ::= (⟨*ns-prefix-decl-qt*⟩ ',')* (⟨*ns-default-decl-qt*⟩ ',')?
(⟨*ns-prefix-decl-qt*⟩ ',')*

⟨*ns-prefix-decl-qt*⟩ ::= 'ns-prefix' ⟨*identifier-qt*⟩ '=' (⟨*IRI*⟩ | ⟨*literal-variable-qt*⟩)

⟨*ns-default-decl-dt*⟩ ::= 'ns-default' (⟨*IRI*⟩ | ⟨*literal-variable-qt*⟩)

⟨*variable-decl-qt*⟩ ::= (('variable' | 'var') ⟨*NCName*⟩ ',')*

⟨*content-qt*⟩ ::= ⟨*literal-content-qt*⟩ | ⟨*comment-qt*⟩ | ⟨*processing-instruction-qt*⟩

⟨*literal-content-qt*⟩ ::= ⟨*String*⟩ | ⟨*literal-variable-qt*⟩
| ⟨*Regexp*⟩

⟨*comment-qt*⟩ ::= 'xcerpt' ':' 'comment' '(' ')' '[' ⟨*literal-content-qt*⟩ ']'

⟨*processing-instruction-qt*⟩ ::= ('xcerpt' ':' 'processing-instruction' | 'xcerpt' ':' 'pi')
'(' 'target-name' '=' ⟨*identifier-qt*⟩ ')' '[' ⟨*literal-content-qt*⟩ ']'

⟨*structured-qt*⟩ ::= ⟨*local-spec-qt*⟩ ⟨*children-list-qt*⟩ ⟨*condition-clause-qt*⟩?

⟨*children-list-qt*⟩ ::= '[' (⟨*query-term*⟩ ',') * ']'
| '{' (⟨*query-term*⟩ ',')* '}'
| '[[' (⟨*query-term*⟩ ',') * ']]'
| '{{' (⟨*query-term*⟩ ',')* '}}'

⟨*condition-clause-qt*⟩ ::= 'where' '(' ⟨*condition-qt*⟩ ')'

⟨*condition-qt*⟩ ::= ⟨*c-parameter*⟩ ⟨*comparison-op*⟩ ⟨*c-parameter*⟩
| ⟨*comparison-op*⟩ '(' ⟨*c-parameter*⟩ ⟨*c-parameter*⟩ ')'
| 'and' '(' ⟨*condition-qt*⟩ ⟨*condition-qt*⟩+ ')'
| 'or' '(' ⟨*condition-qt*⟩ ⟨*condition-qt*⟩+ ')'
| 'not' '(' ⟨*condition-qt*⟩ ')'
| ⟨*c-parameter*⟩

⟨*condition-op*⟩ ::= '==' | '!=' | '<' | '>' | '<=' | '>='

⟨*arithmetic-op*⟩ ::= '+' | '-' | '*' | '/' | '^'

⟨*c-parameter*⟩ ::= ⟨*optional-variable-qt*⟩ | ⟨*variable-qt*⟩
| ⟨*String*⟩ | ⟨*Int*⟩
| ⟨*c-parameter*⟩ ⟨*arithmetic-op*⟩ ⟨*c-parameter*⟩
| ⟨*arithmetic-op*⟩ '(' ⟨*c-parameter*⟩ ⟨*c-parameter*⟩ ')'

⟨*optional-variable-qt*⟩ ::= ⟨*optional-modifier*⟩ ⟨*variable-qt*⟩

⟨*local-spec-qt*⟩ ::= ⟨*term-identifier-qt*⟩? ⟨*ns-label-qt*⟩ ⟨*attr-term-list-qt*⟩

⟨*term-identifier-qt*⟩ ::= ⟨*identifier-qt*⟩ '@'

⟨*ns-label-qt*⟩ ::= (⟨*identifier-qt*⟩ ':')? ⟨*identifier-qt*⟩

⟨*attr-term-list-qt*⟩ ::= '(' (⟨*attr-term-qt*⟩ ','?)* ')'
| '((' (⟨*attr-term-qt*⟩ ','?)* '))'

⟨*attr-term-qt*⟩ ::= ⟨*modified-attr-term-qt*⟩

⟨*modified-attr-term-qt*⟩ ::= ⟨*base-attr-term-qt*⟩
| ⟨*variable-attr-term-qt*⟩
| ⟨*occurrence-modified-attr-term-qt*⟩
| ⟨*selection-modified-attr-term-qt*⟩

⟨*base-attr-term-qt*⟩ ::= ⟨*ns-label-qt*⟩ '=' ⟨*literal-content-qt*⟩

⟨*variable-term-qt*⟩ ::= ⟨*base-term-qt*⟩
| ⟨*term-variable-qt*⟩ ('->' ⟨*base-term-qt*⟩)?

⟨*variable-attr-term-qt*⟩ ::= ⟨*term-variable*⟩ ('->' ⟨*base-attr-term-qt*⟩)?

⟨*variable-qt*⟩ ::= ⟨*term-variable-qt*⟩ | ⟨*literal-variable-qt*⟩

⟨*term-variable-qt*⟩ ::= ⟨*var-specification-qt*⟩

⟨*literal-variable-qt*⟩ ::= ⟨*var-specification-qt*⟩

⟨*var-specification-qt*⟩ ::= ('variable' | 'var')? ⟨*NCName*⟩
| ⟨*anonymous-variable*⟩

⟨*anonymous-variable*⟩ ::= '_'

⟨*selection-modified-qt*⟩ ::= ⟨*selection-modifier*⟩ '(' (⟨*modified-qt*⟩ ',')* ')'

⟨*selection-modified-attr-term-qt*⟩ ::= ⟨*selection-modifier*⟩ '(' (⟨*modified-attr-term-qt*⟩ ',')* ')'

⟨*selection-modifier*⟩ ::= 'except'

⟨*occurrence-modified-qt*⟩ ::= ⟨*occurrence-modifier*⟩ '(' (⟨*modified-qt*⟩ ',')* ')'

⟨*occurrence-modified-attr-term-qt*⟩ ::= ⟨*occurrence-modifier*⟩ '(' (⟨*modified-attr-term-qt*⟩ ',')* ')'

⟨*occurrence-modifier*⟩ ::= ⟨*optional-modifier*⟩ | 'without'

⟨*location-modified-qt*⟩ ::= ⟨*location-modifier*⟩ '(' ⟨*term-variable-qt*⟩ ',')* ')'

⟨*location-modifier*⟩ ::= ⟨*descendant-modifier*⟩ | ⟨*position-modifier*⟩

⟨*descendant-modifier*⟩ ::= 'descendant' | 'desc'

⟨*position-modifier*⟩ ::= ('position' | 'pos') ⟨*number-qt*⟩

⟨*number-qt*⟩ ::= ⟨*Int*⟩ | ⟨*literal-variable-ct*⟩

⟨*top-level-query-term*⟩ ::= ⟨*top-term-level-declare-qt*⟩
| ⟨*optional-top-level-qt*⟩
| ⟨*term-formula-qt*⟩
| ⟨*document-specification-qt*⟩

⟨*top-term-level-declare-qt*⟩ ::= 'declare' ⟨*ns-decl-qt*⟩ ⟨*variable-decl-qt*⟩
'(' ⟨*top-level-construct-term*⟩ ')'

⟨*optional-top-level-qt*⟩ ::= ⟨*optional-modifier*⟩ '(' ⟨*descendant-top-level-qt*⟩ ')'
| ⟨*descendant-top-level-qt*⟩

⟨*descendant-top-level-qt*⟩ ::= ⟨*descendant-modifier*⟩ '(' ⟨*var-restriction-top-level-qt*⟩ ')'
| ⟨*var-restriction-top-level-qt*⟩

⟨*var-restriction-top-level-qt*⟩ ::= ⟨*term-variable*⟩ '->' ⟨*structured-term-qt*⟩
| ⟨*structured-term-qt*⟩

⟨*term-formula-qt*⟩ ::= 'not' '(' ⟨*top-level-query-term*⟩ ')' ⟨*condition-clause-qt*⟩?
| 'and' | 'or' '(' ⟨*top-level-query-term*⟩ ',')? (⟨*top-level-query-term*⟩ ',')+
')' ⟨*condition-clause-qt*⟩?

⟨*document-specification-qt*⟩ ::= 'in' ⟨*xml-document-specification-qt*⟩

⟨*xml-document-specification-qt*⟩ ::= 'xml-document' '((' ⟨*location-qt*⟩
⟨*xml-decl-qt*⟩? ⟨*doctype-decl-qt*⟩? '))' ⟨*xml-document-children-qt*⟩

⟨*location-qt*⟩ ::= 'location' '=' (⟨*IRI*⟩ | ⟨*literal-variable-qt*⟩)

⟨*xml-decl-qt*⟩ ::= ('standalone' '=' ('true' | 'false' | ⟨*literal-variable-qt*⟩))? ('xml-version' '=' ('1.0' | '1.1' | ⟨*literal-variable-qt*⟩))?

⟨*doctype-decl-qt*⟩ ::= ('system-id' '=' ⟨*identifier-qt*⟩)? ('public-id' '=' ⟨*identifier-qt*⟩)? ('root-name' '=' ⟨*identifier-qt*⟩)?

⟨*xml-document-children-qt*⟩ ::= '[[' ⟨*xml-document-content-qt*⟩ ']]'
| '{{' ⟨*xml-document-content-qt*⟩ '}}'
| '{' ⟨*xml-document-content-qt*⟩ '}'
| '[' ⟨*xml-document-content-qt*⟩ ']'

⟨*xml-document-content-qt*⟩ ::= (⟨*annotation-content-qt*⟩ ',')* ⟨*top-level-query-term*⟩ (⟨*annotation-content-qt*⟩ ',')*

⟨*annotation-content-qt*⟩ ::= ⟨*comment-qt*⟩ | ⟨*processing-instruction-qt*⟩

A.5 Programs

⟨*program*⟩ ::= 'PROGRAM' '(' ⟨*goal-block*⟩ ')' 'END'

⟨*goal-block*⟩ ::= ⟨*rule-level-declare-block*⟩* ⟨*goal*⟩ ⟨*rule-level-declare-block*⟩*
| ⟨*rule-level-declare*⟩ '(' ⟨*goal-block*⟩ ')' 'END'

⟨*rule-level-declare-block*⟩ ::= ⟨*goal*⟩ | ⟨*construct-query-rule*⟩ | ⟨*data*⟩
| ⟨*rule-level-declare*⟩ '(' ⟨*rule-level-declare-block*⟩* ')' 'END'

⟨*goal*⟩ ::= 'GOAL' '(' ⟨*out-resource*⟩ ')' 'FROM' '(' ⟨*query-term*⟩ ')' 'END'

⟨*rule*⟩ ::= 'CONSTRUCT' '(' ⟨*construct-term*⟩ ')' 'FROM' '(' ⟨*query-term*⟩ ')' 'END'

⟨*out-resource*⟩ ::= construct-term
| 'out' (⟨*iri*⟩ | ⟨*literal-var*⟩) '(' construct-term ')'

⟨*data*⟩ ::= 'DATA' '(' ⟨*data-term*⟩ ')' 'END'

⟨*rule-level-declare*⟩ ::= 'DECLARE' '(' ⟨*var-decl-qt*⟩* ⟨*ns-decl-qt*⟩* ')'

Appendix B

Grammar for XML-style Term Syntax

B.1 Literal Structures

The literal structures for the XML-style term syntax are identical to the literal structures for non-XML term syntax, i.e., as given in Section A.1.

B.2 Data Terms

⟨*data-term*⟩ ::= ⟨*term-level-declare-dt*⟩ | ⟨*reference-dt*⟩ | ⟨*structured-dt*⟩ | ⟨*content-dt*⟩

⟨*reference-dt*⟩ ::= '^' ⟨*identifier-dt*⟩

⟨*term-level-declare-dt*⟩ ::= '`declare`' ⟨*ns-declaration-dt*⟩ '`(`' (⟨*data-term*⟩ '`,`'?)* '`)`'

⟨*identifier-dt*⟩ ::= ⟨*NCName*⟩ | ⟨*IRI*⟩ | ⟨*String*⟩

⟨*ns-declaration-dt*⟩ ::= (⟨*ns-prefix-declaration-dt*⟩ '`,`'?)* (⟨*ns-default-declaration-dt*⟩ '`,`'?)? (⟨*ns-prefix-declaration-dt*⟩ '`,`'?)*

⟨*ns-prefix-declaration-dt*⟩ ::= '`ns-prefix`' ⟨*identifier-dt*⟩ '=' ⟨*IRI*⟩

⟨*ns-default-declaration-dt*⟩ ::= '`ns-default`' ⟨*IRI*⟩

⟨*content-dt*⟩ ::= ⟨*literal-content-dt*⟩ | ⟨*comment-dt*⟩ | ⟨*processing-instruction-dt*⟩

⟨*literal-content-dt*⟩ ::= ⟨*String*⟩

⟨*comment-dt*⟩ ::= '`<!-`' ⟨*literal-content-dt*⟩ '`->`'

⟨*processing-instruction-dt*⟩ ::= '<?' ⟨*identifier-dt*⟩ ⟨*literal-content-dt*⟩ '?>'

⟨*structured-dt*⟩ ::= '<' ⟨*local-spec-dt*⟩ ⟨*properties-dt*⟩ ('>' ⟨*children-list-dt*⟩ ('</>' | '<' ⟨*ns-label-dt*⟩ '>') | '/>')

⟨*properties-dt*⟩ ::= (‘{’ ‘ordered’ ‘}’)? | (‘{’ ‘unordered’ ‘}’)

⟨*children-list-dt*⟩ ::= ⟨*data-term*⟩*

⟨*local-spec-dt*⟩ ::= ⟨*term-identifier-dt*⟩? ⟨*ns-label-dt*⟩ ⟨*attr-term-list-dt*⟩

⟨*term-identifier-dt*⟩ ::= ⟨*identifier-dt*⟩ ‘@’

⟨*ns-label-dt*⟩ ::= (⟨*identifier-dt*⟩ ‘:’)? ⟨*identifier-dt*⟩

⟨*attr-term-list-dt*⟩ ::= ⟨*attr-term-dt*⟩*

⟨*attr-term-dt*⟩ ::= ⟨*base-attr-term-dt*⟩

⟨*base-attr-term-dt*⟩ ::= ⟨*ns-label-dt*⟩ ‘=’ ⟨*literal-content-dt*⟩

⟨*top-level-data-term*⟩ ::= ⟨*top-term-level-declare-dt*⟩ | ⟨*structured-dt*⟩

⟨*top-term-level-declare-dt*⟩ ::= ‘declare’ ⟨*ns-declaration-dt*⟩ ‘(’ ⟨*top-level-data-term*⟩ ‘)’

- There is an additional restriction on the production for ⟨*structured-dt*⟩: the (namespace, local name) pair used as label in the end element tag and the (namespace, local name) pair used in the start element tag (i.e., produced as part of ⟨*local-spec-dt*⟩) must be (modulo whitespace) component wise equivalent character sequences.

B.3 Construct Terms

⟨*construct-term*⟩ ::= ⟨*term-level-declare-ct*⟩ | ⟨*reference-ct*⟩ | ⟨*structured-ct*⟩ | ⟨*content-ct*⟩
| ⟨*term-variable-ct*⟩
| ⟨*modified-ct*⟩

⟨*reference-ct*⟩ ::= ‘^’ ⟨*identifier-ct*⟩

⟨*term-level-declare-ct*⟩ ::= ‘declare’ ⟨*ns-declaration-ct*⟩ ‘(’ (⟨*construct-term*⟩ ‘,’?)* ‘)’

⟨*identifier-ct*⟩ ::= ⟨*NCName*⟩ | ⟨*IRI*⟩ | ⟨*String*⟩ | ⟨*literal-variable-ct*⟩

⟨*ns-declaration-ct*⟩ ::= (⟨*ns-prefix-declaration-ct*⟩ ‘,’?)* (⟨*ns-default-declaration-ct*⟩ ‘,’?)?
(⟨*ns-prefix-declaration-ct*⟩ ‘,’?)*

⟨*ns-prefix-declaration-ct*⟩ ::= ‘ns-prefix’ ⟨*identifier-ct*⟩ ‘=’ (⟨*IRI*⟩ | ⟨*literal-variable-ct*⟩)

⟨*ns-default-declaration-dt*⟩ ::= ‘ns-default’ (⟨*IRI*⟩ | ⟨*literal-variable-ct*⟩)

⟨*content-ct*⟩ ::= ⟨*literal-content-ct*⟩ | ⟨*comment-ct*⟩ | ⟨*processing-instruction-ct*⟩

⟨*literal-content-ct*⟩ ::= ⟨*String*⟩ | ⟨*literal-variable-ct*⟩

⟨*comment-dt*⟩ ::= ‘<!-’ ⟨*literal-content-ct*⟩ ‘->’

⟨*processing-instruction-dt*⟩ ::= '<?' ⟨*identifier-ct*⟩ ⟨*literal-content-ct*⟩ '?>'

⟨*structured-ct*⟩ ::= '<' ⟨*local-spec-ct*⟩ ⟨*properties-ct*⟩ ('>' ⟨*children-list-ct*⟩ ('</>' | '<' ⟨*ns-label-ct*⟩ '>') | '/>')

⟨*properties-ct*⟩ ::= ('{' 'ordered' '}')? | ('{' 'unordered' '}')

⟨*children-list-ct*⟩ ::= ⟨*construct-term*⟩*

⟨*local-spec-ct*⟩ ::= ⟨*term-identifier-ct*⟩? ⟨*ns-label-ct*⟩ ⟨*attr-term-list-ct*⟩

⟨*term-identifier-ct*⟩ ::= ⟨*identifier-ct*⟩ '@'

⟨*ns-label-ct*⟩ ::= (⟨*identifier-ct*⟩ ':')? ⟨*identifier-ct*⟩

⟨*attr-term-list-ct*⟩ ::= ⟨*attr-term-ct*⟩*

⟨*attr-term-ct*⟩ ::= ⟨*base-attr-term-ct*⟩
| ⟨*term-variable-ct*⟩
| ⟨*modified-attr-term-ct*⟩

⟨*base-attr-term-ct*⟩ ::= ⟨*ns-label-ct*⟩ '=' ⟨*literal-content-ct*⟩

⟨*top-level-construct-term*⟩ ::= ⟨*top-term-level-declare-ct*⟩ | ⟨*structured-ct*⟩

⟨*top-term-level-declare-ct*⟩ ::= 'declare' ⟨*ns-declaration-ct*⟩ '(' ⟨*top-level-construct-term*⟩ ')'

⟨*variable-ct*⟩ ::= ⟨*term-variable-ct*⟩ | ⟨*literal-variable-ct*⟩

⟨*term-variable-ct*⟩ ::= ⟨*var-specification-ct*⟩

⟨*literal-variable-ct*⟩ ::= ⟨*var-specification-ct*⟩

⟨*var-specification-ct*⟩ ::= ('variable' | 'var')? ⟨*NCName*⟩

⟨*modified-ct*⟩ ::= ⟨*grouping-ct*⟩ | ⟨*optional-ct*⟩

⟨*modified-attr-term-ct*⟩ ::= ⟨*grouping-attr-term-ct*⟩ | ⟨*optional-attr-term-ct*⟩

⟨*grouping-ct*⟩ ::= ⟨*grouping-modifier*⟩ '(' (⟨*construct-term*⟩ ',')* ')' ⟨*groupby*⟩? ⟨*orderby*⟩?

⟨*grouping-attr-term-ct*⟩ ::= ⟨*grouping-modifier*⟩ '(' ⟨*attr-term-ct*⟩? ')' ⟨*groupby*⟩? ⟨*orderby*⟩?

⟨*grouping-modifier*⟩ ::= 'all'
| 'some' ⟨*number-ct*⟩
| 'first' ⟨*interval-ct*⟩

⟨*orderby*⟩ ::= 'order-by' '(' ((⟨*optional-variable*⟩ | ⟨*variable-ct*⟩) ',')* ')' ⟨*order-relation*⟩?

⟨*order-relation*⟩ ::= 'ascending' | 'descending' | ⟨*NCName*⟩

⟨*groupby*⟩ ::= 'group-by' '(' ((⟨*optional-variable*⟩ | ⟨*variable-ct*⟩) ',')* ')' ⟨*equivalence-relation*⟩?

⟨*equivalence-relation*⟩ ::= ⟨*NCName*⟩

⟨*optional-variable*⟩ ::= ⟨*optional-modifier*⟩ ⟨*variable-ct*⟩

⟨*interval-ct*⟩ ::= ⟨*number-ct*⟩ '-' ⟨*number-ct*⟩
| ⟨*number-ct*⟩ '-'
| '+'

⟨*number-ct*⟩ ::= ⟨*Int*⟩ | ⟨*literal-variable-ct*⟩

⟨*optional-ct*⟩ ::= ⟨*optional-modifier*⟩ '(' (⟨*construct-subterm*⟩ ',')* ')'
('with-default' '(' (⟨*construct-subterm*⟩ ',')* ')')?

⟨*optional-attr-term-ct*⟩ ::= ⟨*optional-modifier*⟩ '(' (⟨*attr-term-ct*⟩',')* ')'
('with-default' '(' (⟨*attr-term-ct*⟩ ',')* ')')?

⟨*optional-modifier*⟩ ::= 'optional' | 'opt'

B.4 Query Terms

⟨*query-term*⟩ ::= ⟨*modified-qt*⟩
| ⟨*term-level-declare-qt*⟩

⟨*term-level-declare-qt*⟩ ::= 'declare' ⟨*ns-decl-qt*⟩ ⟨*variable-decl-qt*⟩
'('(⟨*query-term*⟩ ',')* ')'

⟨*modified-qt*⟩ ::= ⟨*variable-term-qt*⟩
| ⟨*location-modified-qt*⟩
| ⟨*occurrence-modified-qt*⟩
| ⟨*selection-modified-qt*⟩

⟨*base-term-qt*⟩ ::= ⟨*reference-qt*⟩ | ⟨*content-qt*⟩ | ⟨*structured-qt*⟩

⟨*reference-qt*⟩ ::= '^' ⟨*identifier-qt*⟩

⟨*identifier-qt*⟩ ::= ⟨*NCName*⟩ | ⟨*IRI*⟩ | ⟨*String*⟩
| ⟨*literal-variable-qt*⟩
| ⟨*Regexp*⟩

⟨*ns-decl-qt*⟩ ::= (⟨*ns-prefix-decl-qt*⟩ ',')* (⟨*ns-default-decl-qt*⟩ ',')?
(⟨*ns-prefix-decl-qt*⟩ ',')*

⟨*ns-prefix-decl-qt*⟩ ::= 'ns-prefix' ⟨*identifier-qt*⟩ '=' (⟨*IRI*⟩ | ⟨*literal-variable-qt*⟩)

⟨*ns-default-decl-dt*⟩ ::= 'ns-default' (⟨*IRI*⟩ | ⟨*literal-variable-qt*⟩)

⟨*variable-decl-qt*⟩ ::= (('variable' | 'var') ⟨*NCName*⟩ ',')*

⟨*content-qt*⟩ ::= ⟨*literal-content-qt*⟩ | ⟨*comment-qt*⟩ | ⟨*processing-instruction-qt*⟩

⟨*literal-content-qt*⟩ ::= ⟨*String*⟩ | ⟨*literal-variable-qt*⟩
| ⟨*Regexp*⟩

⟨*comment-qt*⟩ ::= '<!-' ⟨*literal-content-qt*⟩ '->'

⟨*processing-instruction-qt*⟩ ::= '<?' ⟨*identifier-qt*⟩ ⟨*literal-content-qt*⟩ '?>'

⟨*structured-qt*⟩ ::= '<' ⟨*local-spec-qt*⟩ ⟨*properties-qt*⟩ ('>' ⟨*children-list-qt*⟩ ('</>' | '<' ⟨*ns-label-qt*⟩ '>') | '/>') ⟨*condition-clause-qt*⟩?

⟨*properties-qt*⟩ ::= ('{' 'ordered' '}')? | ('{' 'unordered' '}') (('{' 'total' '}')? | '{' 'partial' '}') (('{' 'total attributes' '}')? | '{' 'partial attributes' '}')

⟨*children-list-qt*⟩ ::= ⟨*query-term*⟩*

⟨*condition-clause-qt*⟩ ::= 'where' '(' ⟨*condition-qt*⟩ ')'

⟨*condition-qt*⟩ ::= ⟨*c-parameter*⟩ ⟨*comparison-op*⟩ ⟨*c-parameter*⟩
| ⟨*comparison-op*⟩ '(' ⟨*c-parameter*⟩ ⟨*c-parameter*⟩ ')'
| 'and' '(' ⟨*condition-qt*⟩ ⟨*condition-qt*⟩+ ')'
| 'or' '(' ⟨*condition-qt*⟩ ⟨*condition-qt*⟩+ ')'
| 'not' '(' ⟨*condition-qt*⟩ ')'
| ⟨*c-parameter*⟩

⟨*condition-op*⟩ ::= '==' | '!=' | '<' | '>' | '<=' | '>='

⟨*arithmetic-op*⟩ ::= '+' | '-' | '*' | '/' | '^'

⟨*c-parameter*⟩ ::= ⟨*optional-variable-qt*⟩ | ⟨*variable-qt*⟩
| ⟨*String*⟩ | ⟨*Int*⟩
| ⟨*c-parameter*⟩ ⟨*arithmetic-op*⟩ ⟨*c-parameter*⟩
| ⟨*arithmetic-op*⟩ '(' ⟨*c-parameter*⟩ ⟨*c-parameter*⟩ ')'

⟨*optional-variable-qt*⟩ ::= ⟨*optional-modifier*⟩ ⟨*variable-qt*⟩

⟨*local-spec-qt*⟩ ::= ⟨*term-identifier-qt*⟩? ⟨*ns-label-qt*⟩ ⟨*attr-term-list-qt*⟩

⟨*term-identifier-qt*⟩ ::= ⟨*identifier-qt*⟩ '@'

⟨*ns-label-qt*⟩ ::= (⟨*identifier-qt*⟩ ':')? ⟨*identifier-qt*⟩

⟨*attr-term-list-qt*⟩ ::= ⟨*attr-term-qt*⟩*

⟨*attr-term-qt*⟩ ::= ⟨*modified-attr-term-qt*⟩

⟨*modified-attr-term-qt*⟩ ::= ⟨*base-attr-term-qt*⟩
| ⟨*variable-attr-term-qt*⟩
| ⟨*occurrence-modified-attr-term-qt*⟩
| ⟨*selection-modified-attr-term-qt*⟩

⟨*base-attr-term-qt*⟩ ::= ⟨*ns-label-qt*⟩ '=' ⟨*literal-content-qt*⟩

⟨*variable-term-qt*⟩ ::= ⟨*base-term-qt*⟩
| ⟨*term-variable-qt*⟩ ('->' ⟨*base-term-qt*⟩)?

⟨*variable-attr-term-qt*⟩ ::= ⟨*term-variable*⟩ ('->' ⟨*base-attr-term-qt*⟩)?

⟨*variable-qt*⟩ ::= ⟨*term-variable-qt*⟩ | ⟨*literal-variable-qt*⟩

⟨*term-variable-qt*⟩ ::= ⟨*var-specification-qt*⟩

⟨*literal-variable-qt*⟩ ::= ⟨*var-specification-qt*⟩

⟨*var-specification-qt*⟩ ::= ('variable' | 'var')? ⟨*NCName*⟩
| ⟨*anonymous-variable*⟩

⟨*anonymous-variable*⟩ ::= '_'

⟨*selection-modified-qt*⟩ ::= ⟨*selection-modifier*⟩ '(' (⟨*modified-qt*⟩ ',')* ')'

⟨*selection-modified-attr-term-qt*⟩ ::= ⟨*selection-modifier*⟩ '(' (⟨*modified-attr-term-qt*⟩ ',')* ')'

⟨*selection-modifier*⟩ ::= 'except'

⟨*occurrence-modified-qt*⟩ ::= ⟨*occurrence-modifier*⟩ '(' (⟨*modified-qt*⟩ ',')* ')'

⟨*occurrence-modified-attr-term-qt*⟩ ::= ⟨*occurrence-modifier*⟩ '(' (⟨*modified-attr-term-qt*⟩ ',')* ')'

⟨*occurrence-modifier*⟩ ::= ⟨*optional-modifier*⟩ | 'without'

⟨*location-modified-qt*⟩ ::= ⟨*location-modifier*⟩ '(' ⟨*term-variable-qt*⟩ ',')* ')'

⟨*location-modifier*⟩ ::= ⟨*descendant-modifier*⟩ | ⟨*position-modifier*⟩

⟨*descendant-modifier*⟩ ::= 'descendant' | 'desc'

⟨*position-modifier*⟩ ::= ('position' | 'pos') ⟨*number-qt*⟩

⟨*number-qt*⟩ ::= ⟨*Int*⟩ | ⟨*literal-variable-ct*⟩

⟨*top-level-query-term*⟩ ::= ⟨*top-term-level-declare-qt*⟩
| ⟨*optional-top-level-qt*⟩
| ⟨*term-formula-qt*⟩
| ⟨*document-specification-qt*⟩

⟨*top-term-level-declare-qt*⟩ ::= 'declare' ⟨*ns-decl-qt*⟩ ⟨*variable-decl-qt*⟩
'(' ⟨*top-level-construct-term*⟩ ')'

⟨*optional-top-level-qt*⟩ ::= ⟨*optional-modifier*⟩ '(' ⟨*descendant-top-level-qt*⟩ ')'
| ⟨*descendant-top-level-qt*⟩

⟨*descendant-top-level-qt*⟩ ::= ⟨*descendant-modifier*⟩ '(' ⟨*var-restriction-top-level-qt*⟩ ')'
| ⟨*var-restriction-top-level-qt*⟩

⟨*var-restriction-top-level-qt*⟩ ::= ⟨*term-variable*⟩ '->' ⟨*structured-term-qt*⟩
| ⟨*structured-term-qt*⟩

⟨*term-formula-qt*⟩ ::= 'not' '(' ⟨*top-level-query-term*⟩ ')' ⟨*condition-clause-qt*⟩?
| 'and' | 'or' '(' ⟨*top-level-query-term*⟩ ',')? (⟨*top-level-query-term*⟩ ',')?)+
')' ⟨*condition-clause-qt*⟩?

⟨*document-specification-qt*⟩ ::= 'in' ⟨*xml-document-specification-qt*⟩

⟨*xml-document-specification-qt*⟩ ::= 'xml-document' '((' ⟨*location-qt*⟩
⟨*xml-decl-qt*⟩? ⟨*doctype-decl-qt*⟩? '))' ⟨*xml-document-children-qt*⟩

⟨*location-qt*⟩ ::= 'location' '=' (⟨*IRI*⟩ | ⟨*literal-variable-qt*⟩)

⟨*xml-decl-qt*⟩ ::= ('standalone' '=' ('true' | 'false' | ⟨*literal-variable-qt*⟩))? ('xml-version'
'=' ('1.0' | '1.1' | ⟨*literal-variable-qt*⟩))?

⟨*doctype-decl-qt*⟩ ::= ('system-id' '=' (⟨*String*⟩ | ⟨*literal-variable-qt*⟩))? ('public-id' '='
(⟨*String*⟩ | ⟨*literal-variable-qt*⟩))? ('root-name' '=' (⟨*String*⟩ | ⟨*literal-variable-qt*⟩))?

⟨*xml-document-children-qt*⟩ ::= '[[' ⟨*xml-document-content-qt*⟩ ']]'
| '{{' ⟨*xml-document-content-qt*⟩ '}}'
| '{' ⟨*xml-document-content-qt*⟩ '}'
| '[' ⟨*xml-document-content-qt*⟩ ']'

⟨*xml-document-content-qt*⟩ ::= (⟨*annotation-content-qt*⟩ ',')* ⟨*top-level-query-term*⟩
(⟨*annotation-content-qt*⟩ ',')*

⟨*annotation-content-qt*⟩ ::= ⟨*comment-qt*⟩ | ⟨*processing-instruction-qt*⟩

B.5 Programs

⟨*program*⟩ ::= 'PROGRAM' '(' ⟨*goal-block*⟩ ')' 'END'

⟨*goal-block*⟩ ::= ⟨*rule-level-declare-block*⟩* ⟨*goal*⟩ ⟨*rule-level-declare-block*⟩*
| ⟨*rule-level-declare*⟩ '(' ⟨*goal-block*⟩ ')' 'END'

⟨*rule-level-declare-block*⟩ ::= ⟨*goal*⟩ | ⟨*construct-query-rule*⟩ | ⟨*data*⟩
| ⟨*rule-level-declare*⟩ '(' ⟨*rule-level-declare-block*⟩* ')' 'END'

⟨*goal*⟩ ::= 'GOAL' '(' ⟨*out-resource*⟩ ')' 'FROM' '(' ⟨*query-term*⟩ ')' 'END'

⟨*rule*⟩ ::= 'CONSTRUCT' '(' ⟨*construct-term*⟩ ')' 'FROM' '(' ⟨*query-term*⟩ ')' 'END'

⟨*out-resource*⟩ ::= construct-term
| 'out' (⟨*iri*⟩ | ⟨*literal-var*⟩) '(' construct-term ')'

⟨*data*⟩ ::= 'DATA' '(' ⟨*data-term*⟩ ')' 'END'

⟨*rule-level-declare*⟩ ::= 'DECLARE' '(' ⟨*var-decl-qt*⟩* ⟨*ns-decl-qt*⟩* ')'

Appendix C

Relax NG Schema for XML Syntax

C.1 Parameterized Grammars: Terms, Declarations, Modifiers, etc.

C.1.1 Declarations

```
default namespace = "http://xcerpt.org/ns/core/1.0"
namespace a = "http://relaxng.org/ns/compatibility/annotations/1.0"

start = declare-block

## A declare block with an empty content and both namespace and variable declarations.
declare-block =
  element declare { (ns-declaration | var-declaration)*, content }
ns-declaration =
  ns-prefix-declaration*,
  (element ns-default {
     element iri { iri.class }
   }
   | ns-prefix-declaration),
  ns-prefix-declaration*
ns-prefix-declaration =
  element ns-prefix {
    element name { ncname.class },
    element iri { iri.class }
  }
var-declaration =
  element variable {
    attribute name { xsd:NCName }
  }
content = empty
iri.class |= text
ncname.class |= xsd:NCName
```

C.1.2 Conditions

```
namespace a = "http://relaxng.org/ns/compatibility/annotations/1.0"

start = condition
```

```
## A condition is an opaque expression involving simple arithmetic and comparisons. This is preliminary syntax.
condition = element condition { expression }
expression =
  element arithmetic-expression {
    attribute operator { "+" | "-" | "*" | "/" | "^" },
    expression,
    expression
  }
  | element comparison-expression {
      attribute operator { "eq" | "neq" | "lt" | "gt" | "leq" | "geq" },
      expression,
      expression
    }
  | grammar {
      include "formula.rnc" {
        content = parent expression
      }
    }
  | content
content = empty
```

C.1.3 Formulas

```
default namespace = "http://xcerpt.org/ns/core/1.0"
namespace a = "http://relaxng.org/ns/compatibility/annotations/1.0"

start = formula
formula =
  element and { formula, formula+, condition? }
  | element or { formula, formula+, condition? }
  | element not { formula }
  | content
condition = empty
content = empty
```

C.1.4 Modifiers

```
default namespace = "http://xcerpt.org/ns/core/1.0"
namespace a = "http://relaxng.org/ns/compatibility/annotations/1.0"

start = grouping
content = empty
grouping =
  element all { content, order-by?, group-by? }
  | element some { number, content, order-by?, group-by? }
  | element first { interval, content, order-by?, group-by? }
order-by =
  element order-by {
    attribute order-relation { text }?,
    optional-variable+
  }
group-by =
  element group-by {
    attribute equivalence-relation { text }?,
    optional-variable+
  }
optional-variable =
```

```
  element optional { variable }
  | variable
variable = empty
interval =
  element interval {
    element min { number-literal.class },
    element max { number-literal.class }
  }
number = element number { number-literal.class }
number-literal.class = xsd:int | variable
optional =
  element optional {
    content,
    element with-default { content }?
  }
```

C.1.5 Term

```
default namespace = "http://xcerpt.org/ns/core/1.0"
namespace a = "http://relaxng.org/ns/compatibility/annotations/1.0"

start = top-level-term.class

## A term that may occur at top-level. Slightly more
## restricted than a basic term.
top-level-term.class =
  structured-term
  | grammar {
      include "declare-block.rnc" {
        content = parent top-level-term.class*
        var-declaration = empty
      }
    }

## A generic Xcerpt term. Variants are data, construct, and query terms.
term.class |=
  reference | content-term | structured-term | term-level-declare

## A declaration block on term level allows possibly (in data and construct terms) only namespace declarations.
term-level-declare =
  grammar {
    include "declare-block.rnc" {
      content = parent term.class*
      var-declaration = empty
    }
  }

## A structured term is a term that may have children and
## attributes. It contrasts with literal content.
structured-term =
  element element { term-local-spec, term-children, term-condition? }

## Some terms may have additional constraints attached to them.
term-condition = empty

## The specification of the 'local' properties of a term: identifier, label, namespace, and attributes.
term-local-spec = term-identifier?, ns-label, attr-term-list

## The defining occurrence of a reference, i.e. "id @" in term syntax.
```

```
term-identifier = element identifier { identifier.class }

## Label and namespace of an Xcerpt term or attribute.
ns-label =
  element label {
    element ns { identifier.class }?,
    identifier.class
  }

## A term specifying the attributes of an element.
attr-term-list =
  element attributes {
    attribute total { total.class },
    attribute-term.class*
  }

## Class of values for attributes specifying totality or
## partiality of a term's children or attribute list.
total.class |= "true"

## A attribute term is an attribute possibly modified with respect to location, modality, and selection.
attribute-term.class |= base-attribute

## An attribute consists of a label and an attribute content.
base-attribute =
  element attribute {
    ns-label,
    element value { literal-content.class }
  }

## An identifier such as a namespace or label.
identifier.class |= text
content-term = literal-content.class | annotation-content

## Content kinds that can be used to annotate elements.
annotation-content =
  element comment { literal-content.class }
  | element processing-instruction {
      attribute target { identifier.class },
      literal-content.class
    }

## Character data or other atomic content.
literal-content.class |= text

## The children of a term can be ordered or unordered, total or partial.
term-children =
  element children {
    attribute ordered { "true" | "false" },
    attribute total { total.class },
    term.class*
  }

## The using occurrence of a reference, i.e. "^ id" in term syntax.
reference = element reference { identifier.class }
```

C.2 Grammar for Xcerpt Programs

```
default namespace = "http://xcerpt.org/ns/core/1.0"
namespace a = "http://relaxng.org/ns/compatibility/annotations/1.0"

start = program
# ---------------------------------------------------------------------

# Programs: Rules, Data, and Goals

# ---------------------------------------------------------------------

## An Xcerpt program is a set of (one or more) goals, as well as (any number of) rules and inline data terms (like facts in Prolog). Rules and data terms may be surrounded by declaration blocks.
program = element program { goal-block }
goal-block =
  rule-level-block*
  | goal
  | rule-level-block*
  | grammar {
      include "declare-block.rnc" {
        content = parent goal-block
      }
    }

## Rule-level blocks form the basic block structure of an Xcerpt programs: goals, rules, and inline data terms form the basic block structures. They can be included into declaration blocks that define the scope of variable and namespace declarations.
rule-level-block =
  goal
  | rule
  | data
  |
    ## A declaration block on rule level allows both variable and namespace declarations.
    grammar {
      include "declare-block.rnc" {
        content = parent rule-level-block*
      }
    }

## A rule specifies how from data matched by the query term new data is constructed according to a construct term.
rule =
  element rule {
    element construct { construct-term },
    element from { query-term }
  }

## A goal is a rule, where the resulting data is written to a specified resource. Hence, goals are not chained.
goal =
  element goal {
    element out {
      (variable-ct
       | attribute value {
          text
          >> a:documentation [
               "This should in-fact be a IRI as by RFC 3987. Since XML Schema datatypes only provides the anyURI datatype for URIs conforming the older RFC 2396, arbitrary text is allowed."
             ]
```

```
      }),
    element construct { construct-term }
   },
   element from { query-term }
  }

## An inline data term allows the specification of data terms inside Xcerpt programs similar to facts in Prolog.
data = element data { data-term }
# ---------------------------------------------------------------------------

# Data Terms

# ---------------------------------------------------------------------------
data-term =
 grammar {
   include "term.rnc"
 }
# ---------------------------------------------------------------------------

# Construct Terms

# ---------------------------------------------------------------------------

## Variables for construct terms.
variable-ct =
 element variable {
   attribute name { xsd:NCName }
 }

## A construct term differs from a data term in the addition of variables. As a corollary a few so-called modifiers
   are needed to indicate, e.g., how to group the variables or whether a variable may have no bindings.
construct-term =
 grammar {
   variable-ct = parent variable-ct
   # Add grouping and optional for attributes
   modified-attribute =
    grammar {
     include "modifiers.rnc" {
      start = grouping
      content = parent attribute-term.class
      variable = parent variable-ct
     }
    }
    | grammar {
       include "modifiers.rnc" {
        start = optional
        content = parent attribute-term.class*
        variable = parent variable-ct
       }
      }
   # Add grouping and optional for elements
   modified-term =
    grammar {
     include "modifiers.rnc" {
      start = grouping
```

```
        content = parent term.class*
        variable = parent variable-ct
      }
    }
    | grammar {
        include "modifiers.rnc" {
          start = optional
          content = parent term.class*
          variable = parent variable-ct
        }
      }

  ## Construct terms may also be variables or modified by
  ## grouping and optional modifiers.
  term.class |= variable-ct | modified-term

  ## Construct attribute terms may also be variables or modified by
  ## grouping and optional modifiers.
  attribute-term.class |= variable-ct | modified-attribute
  # Add variables to identifiers and literal content
  identifier.class |= variable-ct
  literal-content.class |= variable-ct
  include "term.rnc"
 }
# − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − −

# Query Terms

# − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − −

## A POSIX.1 regular expression annotated with variables may occur in query terms at the position of identifiers or literal content.
regular-expression =
 element regexp {
   attribute value { text }
 }

## Query terms can evidently become the most complex of the three term kinds in Xcerpt. As construct terms they add variables to data terms. But they also provide means for expressing incompleteness: partial terms, desc and position location modifiers, etc. A construct term differs from a data term in the addition of variables.
query-term =
 grammar {

   ## Variables for query terms.
   variable =
     element variable {
       attribute anonymous { "true" }
       | attribute name { xsd:NCName }
     }
   # #1# TOP−LEVEL QUERY TERM ##
   optional-top-level-term =
     element optional { descendant-top-level-term }
     | descendant-top-level-term
   descendant-top-level-term =
     element descendant { var-restriction-top-level-term }
     | var-restriction-top-level-term
   var-restriction-top-level-term =
```

```
  element restriction { variable, structured-term }
  | structured-term
term-formula =
  grammar {
    include "formula.rnc" {
      content = parent optional-top-level-term
      condition = parent condition-clause
    }
  }
document-specification =
  element xml-document {
    attribute location { text },
    element xml-declaration {
      attribute standalone { "true" | "false" | variable }?,
      attribute xml-version { "1.0" | "1.1" | variable }?
    }?,
    element doctype {
      attribute system-id { identifier.class }?,
      attribute public-id { identifier.class }?,
      attribute root-name { identifier.class }?
    }?,
    element children {
      annotation-content*, top-level-term.class, annotation-content*
    }
  }
# #2# CONDITION CLAUSES ##
condition-clause =
  element condition {
    grammar {
      include "formula.rnc" {
        content = parent comparison*
      }
    }
  }
comparison =
  element comparison {
    attribute operator {
      "eq" | "neq" | "lt" | "bt" | "elt" | "ebt"
    },
    arithmetics,
    arithmetics
  }
  | element optional { comparison }
  | arithmetics
arithmetics =
  element arithmetics {
    attribute operator {
      "plus" | "minus" | "times" | "div" | "power"
    },
    (arithmetics | arithmetic-parameter),
    (arithmetics | arithmetic-parameter)
  }
  | element optional { arithmetics }
  | arithmetic-parameter
arithmetic-parameter =
  variable
  | element value { text }
# #3# Modified terms ##
modified-term =
```

```
  variable-term | location-term | occurrence-term | selection-term
base-term = reference | content-term | structured-term
variable-term =
  base-term
  | variable
  | element restriction { variable, base-term }
location-term =
  element descendant { variable-term }
  | element position {
     element number { variable | xsd:int },
     variable-term
   }
selection-term = element except { modified-term }
occurrence-term =
  element without { modified-term }
  | element optional { modified-term }
# #4# Modified Attribute terms ##
modified-attr-term =
  base-attribute,
  variable-attr-term,
  occurrence-modified-attr-term,
  selection-modified-attr-term
variable-attr-term =
  variable
  | element restriction { variable, base-attribute }
occurrence-modified-attr-term =
  element without { modified-attr-term }
  | element optional { modified-attr-term }
selection-modified-attr-term = element except { modified-attr-term }
# #A1# BASICS ##

# Add variables and regular expressions to identifiers and literal
# content
identifier.class |= variable | parent regular-expression
literal-content.class |= variable | parent regular-expression
include "term.rnc" {
  # Redefine the top-level term for query terms: add variables to
  # declare blocks and allow optional, descendant, variable restriction.
  # Add document specifications
  # Add query term formula

  ## A term that may occur at top-level. Slightly more
  ## restricted than a basic term.
  top-level-term.class =
    optional-top-level-term
    | term-formula
    | document-specification
    | grammar {
       include "declare-block.rnc" {
         content = parent top-level-term.class*
       }
     }
  # Redefine terms: only modified terms, which can in fact be
  # unmodified :-) Term-level declare blocks may also contain variable
  # declarations

  ## A generic Xcerpt term. Variants are data, construct, and query terms.
  term.class = modified-term | term-level-declare

```

```
    ## A declaration block on term level allows possibly (in data and construct terms) only namespace
        declarations.
    term-level-declare =
     grammar {
      include "declare-block.rnc" {
       content = parent term.class*
      }
     }
    # Redefine attributes as well, again to make modification possible

    ## An attribute term is an attribute possibly modified with respect to location, modality, and selection.
    attribute-term.class = modified-attr-term
    # Allow conditions on arbitrary query terms
    term-condition = condition-clause
  }
}
```

C.3 Exemplary Data Term

```
<?xml version="1.0" encoding="UTF-8"?>
<element xmlns="http://xcerpt.org/ns/core/1.0">
  <label>bib</label>
  <attributes total="true" />
  <children ordered="false" total="true">
    <element>
      <identifier>journal.adm</identifier>
      <label>journal</label>
      <attributes total="true" />
      <children ordered="false" total="true">
        <element>
          <label>title</label>
          <attributes total="true" />
          <children ordered="true" total="true"
            >Applied Data Management</children>
        </element>
        <element>
          <label>editors</label>
          <attributes total="true" />
          <children ordered="true" total="true">
            <element>
              <label>editor-in-chief</label>
              <attributes total="true" />
              <children ordered="true" total="true"
                >Titus Pomponius Atticus</children>
            </element>
            <element>
              <label>editor</label>
              <attributes total="true" >
                <attribute><label>region</label><value>Africa</value>
                </attribute>
              </attributes>
              <children ordered="true" total="true"
                >Marcus Aemilius Aemilianus</children>
            </element>
            <element>
              <label>editor</label>
              <attributes total="true" >
                <attribute><label>region</label><value>Gaul</value>
                </attribute>
              </attributes>
              <children ordered="true" total="true"
                >Aulus Hirtius<!-- -->
                <element>
                  <label>affiliation</label>
                  <attributes total="true" />
```

```
47              <children ordered="true" total="true"
                  >Governor, Transalpine Gaul</children>
49            </element>
            </children>
51        </element>
          <element>
            <label>editor</label>
            <attributes total="true" >
55            <attribute><label>region</label><value>Cilicia</value>
              </attribute>
57          </attributes>
            <children ordered="true" total="true"
59            >Marcus Tullius Cicero<!-- -->
              <element>
61              <label>affiliation</label>
                <attributes total="true" />
63              <children ordered="true" total="true">Governor, Cicilia</children>
              </element>
65          </children>
        </element>
67    </children>
    </element>
69  <element>
      <label>publisher</label>
71    <attributes total="true" />
      <children ordered="true" total="true"
73      >Titus Pomponius Atticus</children>
    </element>
75  <element>
      <label>volumes</label>
77    <attributes total="true" />
      <children ordered="true" total="true">
79      <element>
          <identifier>journal.adm.v10</identifier>
81        <label>volume</label>
          <attributes total="true" />
83        <children ordered="true" total="true">
            <element>
85            <identifier>journal.adm.v10.n1</identifier>
              <label>number</label>
87            <attributes total="true">
                <attribute><label>type</label><value>special-issue</value>
89                </attribute>
              </attributes>
91            <children ordered="false" total="true">
                <element>
93                <label>title</label>
                  <attributes total="true" />
95                <children ordered="true" total="true"
                    >Data Processing Challenges in the Age of Wax Tablets</children>
97              </element>
                <element>
99                <label>editorial</label>
                  <attributes total="true" />
101               <children ordered="true" total="true"
                    ><reference>articles.66.cicero.wax</reference></children>
103             </element>
                <element>
105               <label>year</label>
                  <attributes total="true" />
107               <children ordered="true" total="true"
                    >60</children>
109             </element>
                <element>
111               <label>month</label>
                  <attributes total="true" />
113               <children ordered="true" total="true"
                    >july</children>
115             </element>
              </children>
```

```
            </element>
            <element>
              <identifier>journal.adm.v10.n2</identifier>
              <label>number</label>
              <attributes total="true" />
              <children ordered="false" total="true">
                <element>
                  <label>year</label>
                  <attributes total="true" />
                  <children ordered="true" total="true"
                    >60</children>
                </element>
                <element>
                  <label>month</label>
                  <attributes total="true" />
                  <children ordered="true" total="true"
                    >november</children>
                </element>
              </children>
            </element> <!-- number -->
          </children>
        </element> <!-- volume -->
      </children>
    </element> <!-- volumes -->
  </children>
</element> <!-- journal -->

<element>
  <identifier>conf.dmmc</identifier>
  <label>proceedings</label>
  <attributes total="true" />
  <children ordered="false" total="true">
    <element>
      <label>title</label>
      <attributes total="true" />
      <children ordered="true" total="true"
        >Advancements in Data Management for Military and Civil Application</children>
    </element>
    <element>
      <label>editors</label>
      <attributes total="true" />
      <children ordered="true" total="true">
        <element>
          <label>editor</label>
          <attributes total="true" />
          <children ordered="true" total="true"
            >Marcus Aemilius Lepidus<!-- -->
            <element>
              <label>affiliation</label>
              <attributes total="true" />
              <children ordered="true" total="true"
                >Consul, SPQR</children>
            </element></children>
        </element>
        <element>
          <label>editor</label>
          <attributes total="true" />
          <children ordered="true" total="true"
            >Gaius Julius Caesar Octavianus</children>
        </element>
        <element>
          <label>editor</label>
          <attributes total="true" />
          <children ordered="true" total="true"
            >Marcus Antonius</children>
        </element>
      </children>
    </element>
    <element>
      <label>publisher</label>
```

```
      <attributes total="true" />
      <children ordered="true" total="true"
        >SPQR</children>
    </element>
    <element>
      <label>abbrev</label>
      <attributes total="true" />
      <children ordered="true" total="true"
        >DMMC</children>
    </element>
    <element>
      <label>year</label>
      <attributes total="true" />
      <children ordered="true" total="true"
        >44</children>
    </element>
    <element>
      <label>month</label>
      <attributes total="true" />
      <children ordered="true" total="true"
        >july</children>
    </element>
    <element>
      <label>location</label>
      <attributes total="true" />
      <children ordered="true" total="true"
        >Mutina</children>
    </element>
    <element>
      <label>invited-papers</label>
      <attributes total="true" />
      <children ordered="true" total="true">
        <reference>inproc.44.brutus</reference>
        <reference>article.66.scaurus.qumran</reference>
      </children>
    </element> <!-- invited papers -->
  </children>
</element> <!-- proceedings -->

<!-- ////////////////////////////////////////////////////// -->
<element>
  <identifier>article.66.scaurus.qumran</identifier>
  <label>article</label>
  <attributes total="true" />
  <children ordered="false" total="true">
    <element>
      <label>title</label>
      <attributes total="true" />
      <children ordered="true" total="true"
        >From Wax Tablets to Papyri: The Qumran Case Study</children>
    </element>
    <element>
      <label>author</label>
      <attributes total="true" />
      <children ordered="true" total="true"
        >Marcus Aemilius Scaurus<!-- -->
        <element>
          <label>affiliation</label>
          <attributes total="true" />
          <children ordered="true" total="true"
            >Tribun, Gnaeus Pompeius Magnus</children>
        </element>
      </children>
    </element>
    <element>
      <label>in</label>
      <attributes total="true">
        <attribute><label>scrolls</label><value>102-112</value>
          </attribute>
      </attributes>
```

```
      <children ordered="true" total="true">
        <reference>journal.adm.v10.n1</reference>
      </children>
    </element>
    <element>
      <label>citations</label>
      <attributes total="true" />
      <children ordered="true" total="true">
        <element>
          <label>cite</label>
          <attributes total="true">
            <attribute><label>ref</label><value>article.66.cicero.wax</value>
              </attribute>
          </attributes>
          <children ordered="true" total="true" />
        </element>
        <element>
          <label>cite</label>
          <attributes total="true">
            <attribute><label>type</label><value>formatted</value>
              </attribute>
          </attributes>
          <children ordered="true" total="true"
            >M. Aemilius Scaurus (104): A Case for Permanent Storage of
            Senate Proceedings. In: M. Aemilius Scaurus, ed. (104):
            <element>
              <label>i</label>
              <attributes total="true" />
              <children ordered="true" total="true"
                >Princeps Senatus: Honor and Responsibility</children>
            </element>, Chapter 2, 14-88.</children>
        </element> <!-- cite -->
      </children>
    </element> <!-- citations -->
  </children>
</element> <!-- article-->

<element>
  <identifier>article.66.cicero.wax</identifier>
  <label>article</label>
  <attributes total="true" />
  <children ordered="false" total="true">
    <element>
      <label>title</label>
      <attributes total="true" />
      <children ordered="true" total="true"
        >Space- and Time-Optimal Data Storage on Wax Tablets</children>
    </element>
    <element>
      <label>authors</label>
      <attributes total="true" />
      <children ordered="true" total="true">
        <element>
          <label>author</label>
          <attributes total="true" />
          <children ordered="true" total="true"
            >Marcus Tullius Cicero<!-- -->
            <element>
              <label>affiliation</label>
              <attributes total="true" />
              <children ordered="true" total="true"
                >Governor, Cicilia</children>
            </element>
          </children>
        </element>
        <element>
          <label>author</label>
          <attributes total="true" />
          <children ordered="true" total="true"
            >Marcus Aemilius Lepidus<!-- -->
```

```
            <element>
              <label>affiliation</label>
              <attributes total="true" />
              <children ordered="true" total="true"
                >Gens Aemilia</children>
            </element>
          </children>
        </element>
        <element>
          <label>author</label>
          <attributes total="true" />
          <children ordered="true" total="true"
            >Marcus Tullius Tiro<!-- -->
            <element>
              <label>affiliation</label>
              <attributes total="true" />
              <children ordered="true" total="true"
                >Secretary, M. T. Cicero</children>
            </element>
          </children>
        </element>
      </children>
    </element> <!-- authors -->
    <element>
      <label>in</label>
      <attributes total="true">
        <attribute><label>scrolls</label><value>1-94</value>
        </attribute>
      </attributes>
      <children ordered="true" total="true">
        <reference>journal.adm.v10.n1</reference>
      </children>
    </element>
    <element>
      <label>content</label>
      <attributes total="true">
        <attribute><label>type</label><value>xhtml</value>
          </attribute>
      </attributes>
      <children ordered="true" total="true">
        <declare>
          <ns-default><iri>http://www.w3.org/1999/xhtml</iri></ns-default>
          <element>
            <label>body</label>
            <attributes total="true" />
            <children ordered="true" total="true">
              <comment>incomplete due to melted letters on some tablets</comment>
              <element>
                <label>h1</label>
                <attributes total="true">
                  <attribute><label>id</label><value>contributions</value></attribute>
                </attributes>
                <children ordered="true" total="true">Contributions</children>
              </element>
              <element>
                <label>h1</label>
                <attributes total="true" />
                <children ordered="true" total="true"
                  >A History of Data Storage: From Stone to Parchment</children>
              </element>
              <element>
                <label>p</label>
                <attributes total="true" />
                <children ordered="true" total="true"
                  >Despite recent evidence <element>
                    <label>cite</label>
                    <attributes total="true" />
                    <children ordered="true" total="true"
                      ><reference>article.66.scaurus.qumran</reference></children>
                  </element> ... </children>
```

```
</element> <!-- p -->
<element>
  <label>ol</label>
  <attributes total="true" />
  <children ordered="true" total="true">
    <element>
      <label>li</label>
      <attributes total="true" />
      <children ordered="true" total="true">
        <element>
          <label>em</label>
          <attributes total="true" />
          <children ordered="true" total="true">
            <element>
              <label>strong</label>
              <attributes total="true" />
              <children ordered="true" total="true">Homeric</children>
            </element> Age:</children>
        </element><!-- em -->
        ...
      </children>
    </element> <!-- li -->
    <element>
      <label>li</label>
      <attributes total="true" />
      <children ordered="true" total="true">
        <element>
          <label>em</label>
          <attributes total="true" />
          <children ordered="true" total="true"
            >Age pf the <element>
              <label>strong</label>
              <attributes total="true" />
              <children ordered="true" total="true">Kings</children>
            </element>:</children>
        </element><!-- em -->
        ...
      </children>
    </element> <!-- li -->
  </children>
</element> <!-- ol -->
<element>
  <label>h1</label>
  <attributes total="true">
    <attribute><label>id</label><value>tiro</value></attribute>
  </attributes>
  <children ordered="true" total="true">Notae Tironianae</children>
</element> <!-- hi -->
<element>
  <label>img</label>
  <attributes total="true">
    <attribute><label>title</label><value>Tironian et</value></attribute>
    <attribute><label>src</label><value>...</value></attribute>
  </attributes>
  <children ordered="true" total="true" />
</element> <!-- img -->
<element>
  <label>p</label>
  <attributes total="true" />
  <children ordered="true" total="true"
    >As discussed in <element>
      <label>a</label>
      <attributes total="true">
        <attribute><label>href</label><value>#contributions</value></attribute>
      </attributes>
      <children ordered="true" total="true"> ... </children>
    </element></children>
</element> <!-- p -->
<element>
  <label>h1</label>
```

```
467               <attributes total="true">
                    <attribute><label>id</label><value>tachygraphy</value></attribute>
469               </attributes>
                  <children ordered="true" total="true">Challenges for Tachygraphy on Wax</children>
471             </element> <!-- hi -->
                <element>
                  <label>p</label>
                  <attributes total="true" />
475               <children ordered="true" total="true"
                    >Though conditions for writing on wax tablets are adverse
477                 to tachygraphy, systems as described in <element>
                      <label>a</label>
479                   <attributes total="true">
                        <attribute><label>href</label><value>#tiro</value></attribute>
481                   </attributes>
                      <children ordered="true" total="true"> ... </children>
483                 </element></children>
                </element> <!-- p -->
485           </children>
            </element> <!-- html -->
487       </declare>
        </children>
489     </element> <!-- content -->
    </children>
491 </element> <!-- article-->

493 <element>
    <identifier>inproc.44.brutus</identifier>
495 <label>inproceedings</label>
    <attributes total="true" />
497 <children ordered="false" total="true">
      <element>
499     <label>title</label>
        <attributes total="true" />
501     <children ordered="true" total="true"
          >Efficient Management of Rapidly Changing Personal Records</children>
503   </element>
      <element>
505     <label>authors</label>
        <attributes total="true" />
507     <children ordered="true" total="true">
          <element>
509         <label>author</label>
            <attributes total="true" />
511         <children ordered="true" total="true"
              >Marcus Antonius<!-- -->
513           <element>
                <label>affiliation</label>
515             <attributes total="true" />
                <children ordered="true" total="true"
517               >Consul, SPQR</children>
              </element>
519         </children>
          </element>
521       <element>
            <label>author</label>
523         <attributes total="true" />
            <children ordered="true" total="true"
525           >Decimus Junius Brutus<!-- -->
              <element>
527             <label>affiliation</label>
                <attributes total="true" />
529             <children ordered="true" total="true"
                  >Governor, Cisalpine Gaul</children>
531           </element>
            </children>
533       </element>
        </children>
535   </element> <!-- authors -->
      <element>
```

```
      <label>in</label>
      <attributes total="true">
        <attribute><label>scrolls</label><value>24-48</value>
        </attribute>
      </attributes>
      <children ordered="true" total="true">
        <reference>conf.dmmc</reference>
      </children>
    </element>
    <element>
      <label>content</label>
      <attributes total="true">
        <attribute><label>type</label><value>docbook</value>
        </attribute>
      </attributes>
      <children ordered="true" total="true">
        <declare>
          <ns-default><iri>http://example.org/ns/docbook/simplified/1.0</iri></ns-default>
          <element>
            <label>section</label>
            <attributes total="true" />
            <children ordered="true" total="true">
              <element>
                <label>info</label>
                <attributes total="true" />
                <children ordered="true" total="true">
                  <element>
                    <label>title</label>
                    <attributes total="true" />
                    <children ordered="true" total="true">Introduction</children>
                  </element>
                </children>
              </element>
              <element>
                <label>section</label>
                <attributes total="true" />
                <children ordered="true" total="true">
                  <element>
                    <label>info</label>
                    <attributes total="true" />
                    <children ordered="true" total="true">
                      <element>
                        <label>title</label>
                        <attributes total="true" />
                        <children ordered="true" total="true">Contributions</children>
                      </element>
                    </children>
                  </element>
                  <element>
                    <label>para</label>
                    <attributes total="true" />
                    <children ordered="true" total="true"
                      >The most notable contributions of this article
                      include:<element>
                        <label>list</label>
                        <attributes total="true">
                          <attribute><label>type</label><value>ordered</value></attribute>
                        </attributes>
                        <children ordered="true" total="true">
                          <element>
                            <label>item</label>
                            <attributes total="true" />
                            <children ordered="true" total="true">
                              <element>
                                <label>para</label>
                                <attributes total="true" />
                                <children ordered="true" total="true">
                                  <element>
                                    <label>em</label>
                                    <attributes total="true" />
```

```
                                <children ordered="true" total="true">Clear Evidence</children>
                              </element> of the need ...</children>
                            </element> <!-- para -->
                          </children>
                        </element> <!-- item -->
                        <element>
                          <label>item</label>
                          <attributes total="true" />
                          <children ordered="true" total="true">
                            <element>
                              <label>para</label>
                              <attributes total="true" />
                              <children ordered="true" total="true">A new
                                <element>
                                  <label>em</label>
                                  <attributes total="true" />
                                  <children ordered="true" total="true">methodology</children>
                                </element> to ..., cf. <element>
                                  <label>pageref</label>
                                  <attributes total="true">
                                    <attribute><label>idref</label><value>inproc.44.brutus.s1</value></attribute>
                                  </attributes>
                                  <children ordered="true" total="true" />
                                </element></children>
                            </element> <!-- para -->
                            <element>
                              <label>figure</label>
                              <attributes total="true" />
                              <children ordered="true" total="true">
                                <element>
                                  <label>title</label>
                                  <attributes total="true" />
                                  <children ordered="true" total="true">Chart of Desertions</children>
                                </element>
                                <element>
                                  <label>img</label>
                                  <attributes total="true" />
                                  <children ordered="true" total="true"> ... </children>
                                </element>
                              </children>
                            </element> <!-- figure -->
                            <element>
                              <label>para</label>
                              <attributes total="true" />
                              <children ordered="true" total="true"
                                >As <element>
                                  <label>cite</label>
                                  <attributes total="true" />
                                  <children ordered="true" total="true">
                                    <reference>article.66.cicero.wax</reference>
                                  </children>
                                </element> of the need ...</children>
                            </element> <!-- para -->
                          </children>
                        </element> <!-- item -->
                      </children>
                    </element> <!-- list -->
                  </children>
                </element> <!-- para -->
              </children>
            </element> <!-- section -->
          </children>
        </element> <!-- section -->
        <element>
          <identifier>inproc.44.brutus.s1</identifier>
          <label>section</label>
          <attributes total="true" />
          <children ordered="true" total="true">
            <element>
              <label>info</label>
```

```
                <attributes total="true" />
                <children ordered="true" total="true">
                  <element>
                    <label>title</label>
                    <attributes total="true" />
                    <children ordered="true"
                      total="true">Acknowledgements</children>
                  </element>
                </children>
              </element> <!-- info -->
              <element>
                <label>para</label>
                <attributes total="true" />
                <children ordered="true" total="true"
                  >We would like to thank the editors of <element>
                    <label>cite</label>
                    <attributes total="true" />
                    <children ordered="true" total="true">
                      <reference>journal.adm.v10.n1</reference>
                    </children>
                  </element> ...
                </children>
              </element> <!-- para -->
            </children>
          </element> <!-- section -->
        </declare>
      </children>
    </element> <!-- content -->
  </children>
 </element> <!-- inproceedings-->
</children>
</element>
```

Index

Bibliography

[1] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wienerm. The Lorel Query Language for Semistructured Data. *Int. Journal on Digital Libraries*, 1(1):68–88, 1997.

[2] K. Apt, H. Blair, and A. Walker. Towards a Theory of Deductive Knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, chapter 2, pages 89–148. Morgan Kaufmann, 1988.

[3] J. W. Backus, F. L. Bauer, J. Green, C. Katz, J. McCarthy, A. J. Perlis, H. Rutishauser, K. Samelson, B. Vauquois, J. H. Wegstein, A. van Wijngaarden, and M. Woodger. Revised Report on the Algorithm Language ALGOL 60. *Communications of the ACM*, 6(1):1–17, 1963.

[4] J. Bailey, F. Bry, T. Furche, and S. Schaffert. Web and Semantic Web Query Languages: A Survey. In J. Maluszinsky and N. Eisinger, editors, *Reasoning Web Summer School 2005*. Springer-Verlag, 2005.

[5] P. V. Biron and A. Malhotra. XML Schema Part 2: Datatypes Second Edition. Recommendation, W3C, 2004.

[6] S. Boag, D. Chamberlin, M. F. Fernández, D. Florescu, J. Robie, and J. Siméon. XQuery 1.0: An XML Query Language. Working draft, W3C, 2005.

[7] T. Bray, D. Hollander, and A. Layman. Namespaces in XML. Recommendation, W3C, 1999.

[8] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible Markup Language (XML) 1.0 (Third Edition). Recommendation, W3C, 2004.

[9] L. Braz. Visual Syntax Diagrams for Programming Language Statements. In *Proc. Int. Conf. on Systems Documentation*, pages 23–27. ACM Press, 1990.

[10] F. Bry, T. Furche, L. Badea, C. Koch, S. Schaffert, and S. Berger. Querying the Web Reconsidered: Design Principles for Versatile Web Query Languages. *Journal of Semantic Web and Information Systems*, 1(2), 2005.

[11] F. Bry, T. Furche, S. Schaffert, and A. Schröder. Simulation Unification. Deliverable I4-D5, REWERSE, 2005.

[12] J. Clark. Associating Style Sheets with XML Documents, Version 1.0. Recommendation, W3C, 1999.

[13] J. Clark. XSL Transformations, Version 1.0. Recommendation, W3C, 1999.

[14] J. Clark. RELAX NG Compact Syntax. Committee specification, OASIS, 2002.

[15] J. Clark and M. Murata. RELAX NG Specification. Committee specification, OASIS, 2001.

[16] J. Cowan and R. Tobin. XML Information Set (2nd Ed.). Recommendation, W3C, 2004.

[17] D. Crocker and P. Overell. Augmented BNF for Syntax Specifications: ABNF. Request for Comment (RFC) 2234, IETF, 1997.

[18] S. DeRose, E. Maier, and D. Orchard. XML Linking Language (XLink) Version 1.0. Recommendation, W3C, 2001.

[19] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. A Query Language for XML. In *Proc. Int. World Wide Web Conf.*, 1999.

[20] A. Dovier, C. Piazza, and A. Policriti. An efficient Algorithm for Computing Bisimulation Equivalence. *Theoretical Computer Science*, 311(1-3):221–256, 2004.

[21] M. Duerst and M. Suignard. Internationalized Resource Identifiers (IRIs). RFC (Request for Comments) 3987, IEEE, 2005.

[22] D. C. Fallside and P. Walmsley. XML Schema Part 0: Primer Second Edition. Recommendation, W3C, 2004.

[23] M. Fernández, A. Malhotra, J. Marsh, M. Nagy, and N. Walsh. XQuery 1.0 and XPath 2.0 Data Model. Working draft, W3C, 2005.

[24] R. Gentilini, C. Piazza, and A. Policriti. From Bisimulation to Simulation: Coarsest Partition Problems. *Journal of Automated Reasoning*, 31(1):73–103, 2003.

[25] J. Gosling, B. Joy, G. Steele, and G. Bracha. *Java Language Specification, Third Edition*. Addison-Wesley Professional, 3rd edition, 2005.

[26] T. A. Group. Ieee standard 1003.1, 2004 edition (aka posix.1). IEEE Standard 1003.1, IEEE, The Open Group, 2001-2004.

[27] M. R. Henzinger, T. A. Henzinger, and P. W. Kopke. Computing Simulations on Finite and Infinite Graphs. In *Proc. Symp. on Foundations of Computer Science (FOCS)*, page 453, Washington, DC, USA, 1995. IEEE Computer Society.

[28] ISO/IEC. *ISO/IEC 14977:1996, Syntactic Metalanguage – Extended BNF*, 1996.

[29] M. Kay, N. Walsh, H. Zongaro, S. Boag, and J. Tong. XSLT 2.0 and XQuery 1.0 Serialization. Working draft, W3C, 2005.

[30] A. Malhotra, J. Melton, and N. Walsh. XQuery 1.0 and XPath 2.0 Functions and Operators. Working draft, W3C, 2005.

[31] J. Marsh. XML Base. Recommendation, W3C, 2001.

[32] J. Marsh, D. Veillard, and N. Walsh. xml:id Version 1.0. Recommendation, W3C, 2005.

[33] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: a database management system for semistructured data. *SIGMOD Record*, 26(3):54–66, 1997.

[34] R. Milner. An Algebraic Definition of Simulation Between Programs. In *Proc. Int. Joint Conf. on Artificial Intelligence*, pages 481–489, 1971.

[35] Object Management Group. UML 2.0 Superstructure Specification. Specification, Object Management Group, 2005.

[36] T. Przymusinsik. On the Declarative Semantics of Deductive Databases and Logic Programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, chapter 5, pages 193–216. Morgan Kaufmann, 1988.

[37] S. Schaffert. *Xcerpt: A Rule-Based Query and Transformation Language for the Web*. Dissertation/Ph.D. thesis, University of Munich, 2004.

[38] S. Schaffert, F. Bry, and T. Furche. Initial Draft of a Possible Declarative Semantics for the Language. Deliverable I4-D4, REWERSE, 2005.

[39] M. M. Zloof. Query By Example. In *AFIPS National Computer Conference*, 1975.