University of California
Los Angeles

Parser for Relaxation-Enabled XMLQuery Language (RLXQuery)

Advisor: Wesley W. Chu

Student: Eric Sung

Fall, 2003

Abstract
Query relaxation for XML (eXtensible Markup Language) [4] data is more desired because the data structure in XML model is substantial bigger than in relational model. The XML query relaxation proposed by Dongwon Lee [6] tries to do query approximating using *XML Type Abstraction Hierarchy* (X-TAH). In order implement Lee's proposal, we create the design of RLXQuery Engine [7].

The focus of this paper is RLXQuery parser. The RLXQuery parser is one of three important components to be built in RLXQuery Engine. The job of RLXQuery parser is to parse the user's RLXQuery query string and returns a XQR_RLX class object, which contains relaxation information of the query. If the XQuery part of the RLXQuery return empty result, RLXQuery engine will relax the expressions stored in XQR_RLX class based on the relaxation information and generate an expression-relaxed XQuery.

1. Introduction

Query relaxation technique has been used in relational databases [5][7][8][9], and has proven to be a valuable technique for deriving approximate answers. In our previous work on query relaxation in CoBase [5] project, we extended SQL to CoSQL6y 8. CoSQL provides relaxation operation and control for relational data model.

Increasingly, XML is considered the information exchange format of choice on the Internet, and it is natural that queries among applications should be expressed as queries against data in XML format. This use gives rise to a requirement for a query language designed expressly for XML data sources. In October 1999, W3C formed a group for the purpose of designing such a query language called XQuery [1].

Query relaxation is more important for the XML model than the relational model because unlike in the relational model where users are given a relatively small-sized schema to ask queries, the schema in the XML model is substantially bigger and more complex. Therefore, it is often unrealistic for users to understand the full schema and compose very complex queries at once, and it becomes critical to be able to relax the user's query when the original query yields null or not sufficient answers.

In addition, as the number of data sources available on the web increases, it becomes common to build systems where data are gathered from the heterogeneous data source, where the structures of the participating data source are different although they are using the same ontologies about the same contents. Therefore, the capability to query against differently structured data sources becomes more important.

Dongwon Lee suggested the approach of XML query relaxation by introducing X-TAH and RLXQuery. However, there is no system implementation for Lee's research. Thus, RLXQuery Engine is developed to put Lee's research into real system. Relaxation-Enabled XMLQuery Language (RLXQuery) is the query language used by the engine. RLXQuery is a subset of XQuery to exclude parts of un-used XQuery grammar and an extension of XQuery to add XML query relaxation constructs. There are three major components to be built in RLXQuery Engine: parser, X-TAH manager, and relaxation

kernel. This paper is to focus on the design and development of RLXQuery parser**.** RLXQuery parser can be further divided into three subprojects: ENBF, parser, and XQR_RLX class. **[Figure 1]**

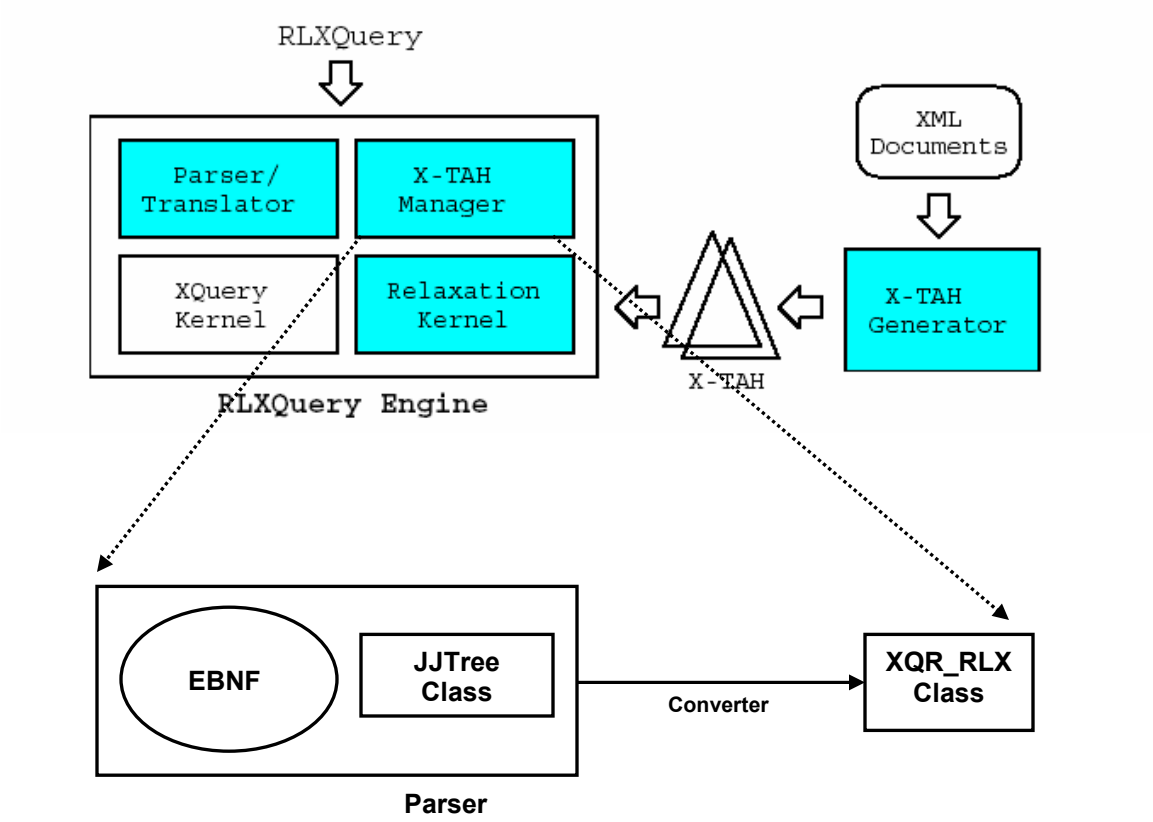


Figure 1: RLXQuery Engine Dataflow Diagram and RXLQuery Parser

2. XQuery

As increasing amounts of information are stored, exchanged, and presented using XML, the ability to intelligently query XML data sources becomes increasingly important. One of the great strengths of XML is its flexibility in representing many different kinds of information from diverse sources. To exploit this flexibility, XQuery is created with the motivation of providing features for retrieving and interpreting information from these diverse sources.

2.1 History

XQuery is designed to meet the requirements identified by the W3C XML Query Working Group [1]. The Query Working Group has identified a requirement for both a human-readable query syntax and an XML-based query syntax. XQuery is derived from an XML query language called Quilt [15], which in turn borrowed features from several other languages, including XPath 1.0 [10], XQL [11], SQL [12]. The first version of XQuery is born in June 2001. After 3 iteration within 2 years, the latest version is released on August 2003 on which our project is based.

2.2 Grammar

Backus-Naur Form (BNF) is a formal mathematical way to describe a language. It is used to formally define the grammar of a language, so that there is no disagreement or ambiguity as to what is allowed and what is not. In fact, BNF is so unambiguous that there is a lot of math theory around these kinds of grammars, and one can actually mechanically construct a parser for a language given a BNF grammar for it.

In a BNF grammar, each non-terminal is described as choice of zero or more sequences of zero or more terminals and non-terminals. Extended Backus-Naur Form EBNF extends BNF with looping, optional parts, and allows choices anywhere, not just at the top level. XQuery grammar is described in EBNF.

2.3 Query Relaxation

Query relaxation is the process of understanding the semantic context and intent of a user query and massaging the query constraints into "near" values that provide "best-fit" answers. Relaxation is a knowledge-based approach to query answering that can provide counter-intuitive and over zealous responses when applied in an uncontrolled manner. A *relaxation mediator* provides operations such as approximately-to, similar-to, or near-to on specific data schemas and/or types. In addition, a relaxation mediator searches for approximate answers automatically whenever a user query returns empty result. Using external knowledge sources, the mediator answers a query by applying controlled rewriting and relaxation of query terms such that the input query is answered to a high level of accuracy and interpretation.

Our specific relaxation mediators use a knowledge structure termed *Type Abstraction Hierarchy* (TAH) to assist in approximately answering queries. Furthermore, we allow the input query to be annotated with control parameters to help guide the mediator in the application of query relaxation.

3. Implementation and Experience

The parser for XQuery relaxation is implemented using Java and JavaCC. The parser is supposed to parse an XQuery relaxation query and populate the XQR_RXL class object representing the RLXQuery. The XQR_RXL class will be then used by relaxation mediator for relaxing the query constrains and producing post-relaxed XQuery.

3.1 RLXQuery: Relaxation-Enabled XML Query Language

CoSQL is our previous work to extend SQL language for relational data model. With the requirement of developing a set of relaxation operation and controls to support query relaxation for the XML model, RLXQuery, a query language that supports the query relaxation based on XQuery, is developed. The major features of the RLXQuery are the following:

- Based on standard XQuery query statements, and downward compatible with the corresponding portion of XQuery FLWR statements.

- Allows the use of a standard XQuery query when there are no sufficient answers for the original query, the system relaxes the query conditions, based on the pre-specified default strategy.

- Allows the user to specify relaxation constructs (e.g. approximate values or conceptual terms, and preference list for certain query condition) in a query

- Allows the user to specify relaxation control constructs such as an unacceptable list for certain query condition, relaxation order for multiple relaxable conditions, minimum answer set size etc. and allows the user to rank the XML answer sets based on the similarity metric specified in the query

3.1.1 Grammar

RLXQuery grammar is described in EBNF. RLXQuery language is a subset of XQuery language with a query relaxation extension. The full RLXQuery EBNF is in **Appendix I**. In this session, we will discuss the RLXQuery EBNF extended from XQuery grammar in our project. Any query can't be described by RLXQuery grammar is considered as an invalid query.

3.2 Java Compiler Compiler

Java Compiler Compiler(JavaCC) [13] is the parser generator for use with Java applications. JavaCC is used by the W3C's XML Query working group to build and test versions of the XQuery grammar [2]. JavaCC reads a grammar specification and converts it to a Java program that can recognize matches to the grammar. In addition to the parser generator itself, JavaCC provides other standard capabilities related to parser generation such as tree building (via a class called JJTree included with JavaCC), actions, debugging, etc.

There are many grammars of various languages like Java, C, and C++ created for JavaCC etc. You can download those grammars from the JavaCC grammar repository on our cobase project web site [14].

3.2.1 JavaCC RLXQuery grammar

The way JavaCC implement XQuery relaxation grammar is also a XML file. The description of the grammar in this file is written in a notation that's very similar to EBNF, so that it's generally fairly easy to translate from one to the other. (The notation has a syntax of its own, making it expressable in JavaCC.)

The main difference between a JavaCC xml file grammar and standard EBNF is that, with the JavaCC version, there are 2 main parts of specifying a grammar: token and product.

3.2.2 Token

Tokens define the terminals of the grammar.
For example, terminal in RLXQuery

NonRelaxable ::= “!”

is equivalent to the following code:

```
<g:token name="NonRelaxable">
  <g:string>!</g:string>
</g:token>
```

3.2.3 Production

Products define the non-terminals of the grammar.
For example, non-terminal in RLXQuery

PredicatePathExpr	::=	(("!")? ("/" | "//"))? PredicateRelativePathExpr

is equivalent to the following code:

```
<g:production name="PredicatePathExpr" if="xquery core">
  <g:optional>
    <g:choice>
        <g:optional>
          <g:ref name="NonRelaxableEdge" />
        </g:optional>
        <g:ref name="Slash" />
        <g:ref name="SlashSlash" />
    </g:choice>
  </g:optional>
  <g:ref name="PredicateRelativePathExpr" />
</g:production>
```

3.2.4 JJTree

JJTree is a JavaCC's companion tool. JJTree is set up to emit a parser whose main job at runtime is not to execute embedded Java actions, but to build an independent parse-tree representation of the expression that's being parsed. This lets you capture the state of the parse session in a single tree that's easy to walk and interrogate at runtime, independent of the parsing code that produced it. Working with a parse tree representation also makes debugging easy and speeds development time.

JJTree is a preprocessor, and generating a parser for a particular BNF. Every node in JJTree is either a terminal or a non-terminal. The class name for the node is SimpleNode. The SimpleNode class provides methods for accessing, setting, and navigating. For example, dump() method output a straightforward, textual representation of the tree for debugging purpose. JjtGetChild(int) let you navigate downward through the parse tree. **Appendix III** and **Appendix IV** shows the JJTree dumps of two RLXQuery queries which are used to verify the correctness of the parser.

JJTree is just an internal representation of JavaCC's parse-tree. However, our RLXQuery engine needs a more complex class XQR_RLX for relaxation process. So, a pre-process is required to convert the tree from JJTree to XQR_RLX.

3.2.5 Top-down vs. bottom-up

The big difference between Yacc, Bison, and JavaCC is that Yacc and Bison work bottom-up, whereas JavaCC works top-down. This means that JavaCC has to make its choices prior to consuming any of the tokens associated with the choice. However, JavaCC's lookahead capabilities allow it to peek well ahead in the token stream without consuming any tokens; the lookahead capabilities ameliorate most of the disadvantages of the top-down approach. Yacc and Bison require BNF grammars, whereas JavaCC accepts EBNF grammars.

Top-down parsing techniques are attractive because of their simplicity, and can often achieve good performance in practice. However, with a top-down parser like JavaCC for XQuery relaxation, left-recursive and top-down unique identifier are 2 most commonly problems occurs.

3.2.6 Left-recursive

Left-recursion is when a non-terminal contains a recursive reference to itself that is not preceded by something that will consume tokens. The parser class produced by JavaCC work with recursive descent. Left-recursion is banned to prevent the generated subroutines from calling themselves recursively ad-infinitum.

Consider the following obviously left recursive production

```
<g:production name="SimpleTerm">
    <g:ref name="SimpleFactor" />
    <g:ref name="Multiply" />
    <g:ref name="SimpleFactor" />
</g:production>

<g:production name="SimpleFactor">
    <g:ref name="SimpleTerm" />
</g:production>
```

Now if the condition is ever true, we have an infinite recursion. Luckly JavaCC will produce an error message, if you have left-recursive productions.
The left-recursive production above can be transformed, using looping, to
or, using right-recursion, to a new production. General methods for left-recursion removal can be found in any text book on compiling.

3.2.7 Top-down non-unique identifier

Some of JavaCC's most common error messages go something like this

Warning: Choice conflict ... Consider using a lookahead of 2 for ...

Read the message carefully. Understand why there is a choice conflict (choice conflicts will be explained shortly) and take appropriate action. In EBNF production like the following:

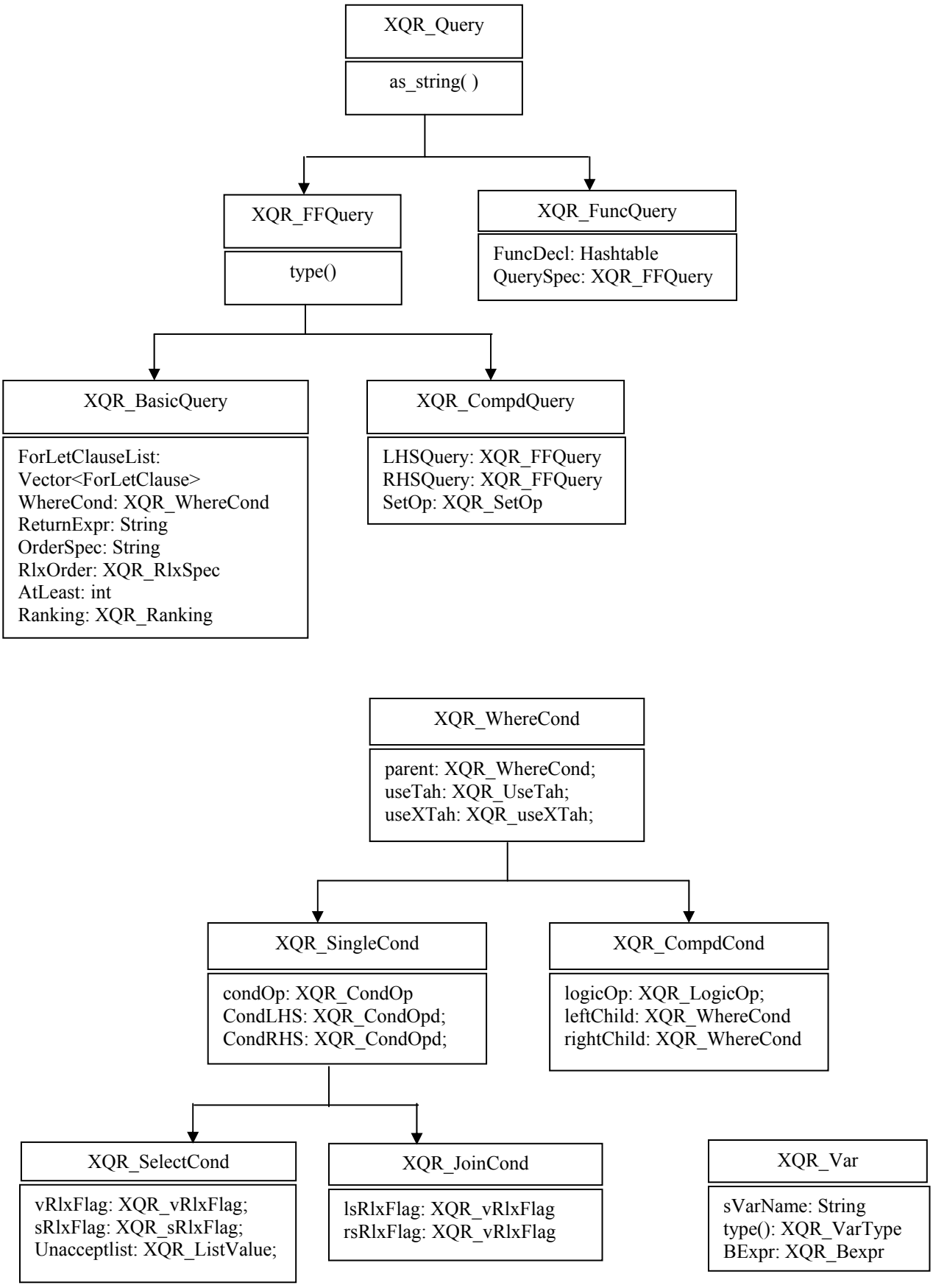
```
<g:production name="SimpleTerm" if="xquery core">
    <g:ref name="SimpleFactor" />
    <g:ref name="Multiply" />
    <g:choice>
      <g:ref name="IntegerLiteral" />
      <g:ref name="NumberLiteral" />
    </g:choice>
</g:production>
```

When the parser applies this production, it must choose between expanding it to IntegerLiter or NumberLiteral. But if the next token is an integer then either choice is appropriate. So you have a "choice conflict". For alternation, the default choice is the first choice; that is, if you ignore the warning, the first choice will be preferred, and in this example, the second statement is unreachable. Another way to resolve conflicts is to rewrite the grammar. The above 2 non-terminals, IntegerLiteral and NumberLiteral can be rewritten to have a unique string token to resolve the ambiguity. For example, if we change to:

```
<g:production name="SimpleTerm" if="xquery core">
    <g:ref name="SimpleFactor" />
    <g:ref name="Multiply" />
    <g:choice>
      <g:sequence>
        <g:ref name="PoundSign" />
        <g:ref name="IntegerLiteral" />
      </g:sequence>
      <g:sequence>
          <g:ref name="SemiColon" />
          <g:ref name="NumberLiteral" />
      </g:sequence>
    </g:choice>
</g:production>
```

then the string “#1234” will be parsed as IntegerLiteral, while “;1234” will be parsed as NumberLiteral.

3.3 XQR_RLX class



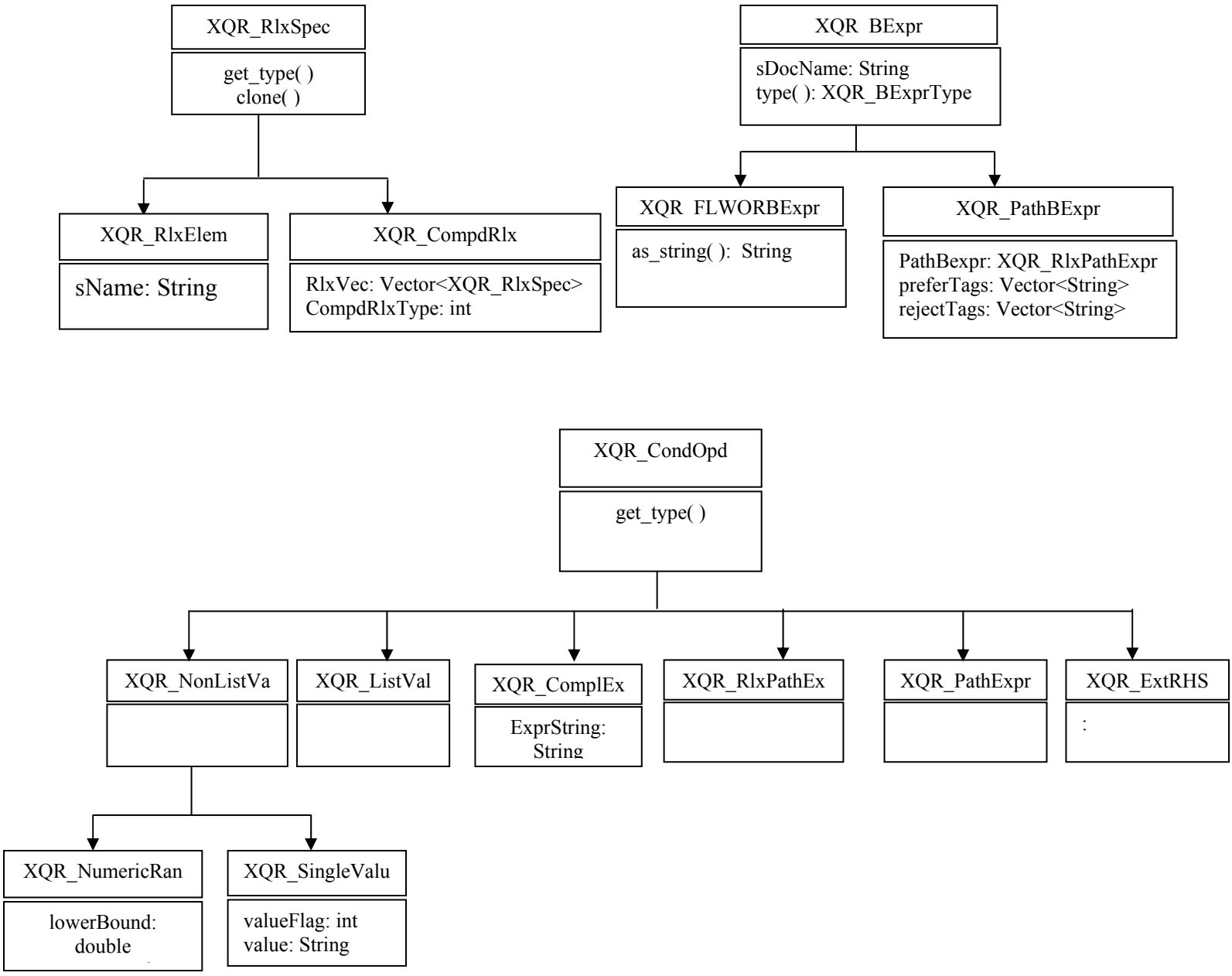


Figure 2: XQR_RLX class digram

XQR_RLX class is the object representation of post-processed JJTree. XQR_RLX is designed with the idea of simplifying the methods of retrieving and re-generating the query relaxation information. Thus, RLXQuery Engine can easily locate the relaxation operation and control of the query. RLXQuery Engine can farther replace the relaxation operation and control with relaxed XQuery expression within the class. **Figure 2** is the class diagram of XQR_RLX. The job of the converter is to take a JJTree class object as input and return a XQR_RLX class object as output.

5. Future works

Two other components: X-TAH manager and relaxation kernel are still in the process of implementation by other members in the group.

6. Acknowledgement

The research and development effort is a team effort. I would like to acknowledge Professor Wesley W. Chu, Sharong Liu, and Dongwon Lee for their contributions in design and implementation.

7. Challenges

The first challenge is to design RXLQuery grammar which works under original XQuery grammar without conflicts. RLXQuery grammar needs to support both relaxation constructs and XQuery grammar. The direct translation from grammar to EBNF doesn't work most of the time. The most common issues in translating grammar to EBNF are recursion/loops, false-positive, and non-unique identifier. Recursion/loops cause the parser never reaching a terminate state. False-positive make the parser accept query with invalid grammar. Non-unique identifier allows non-reaching states in the parser.

The 46 classes in XQR_RLX class architecture are very complex. To make sure that every possible RXLQuery query can be represented in XQR_RLX class hierarchy is a tedious development process. The 46 classes will require tremendous amount of time maintenance if we need to debug or change design in the future.

8. Summary

XML data model becomes more popular for information exchange and its data structure is quite different from the traditional relational data model. XQuery is the query language for XML data model developed to match the power of SQL for relational model. UCLA's CoBase project previously has developed the SQL relaxation engine. Dongwon Lee's Ph.D dissertation proposed the design of RLXQuery engine used for XQuery relaxation, but the implementation is not put into action.

My work in the project is to help RLXQuery Engine's design and development in the areas of RLXQuery EBNF, parser, and XQR_RLX class converter. The RLXQuery parser, implemented in java, parse the RLXQuery string and generate a JJTree object representation of the query. Then, XQR_RLX converter will convert the JJTree into RLXQuery object which is executed later by XQuery kernel. The parser and converter are the major components for RLXQuery Engine. This system will be used to evaluate our proposed XML query relaxation methodology.

Reference

[1] XQuery 1.0 An XML Query Language: http://www.w3.org/TR/2003/WD-xquery-20030822/

[2] XQuery 1.0 Grammar Test Page:
http://www.w3.org/2003/05/applets/xqueryApplet.html

[3] Sharong Liu. ITR: Query Relaxation for XML Database, 2002

[4] Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation (6 October 2000), see http://www.w3.org/TR/REC-xml.
[5] Wesley W. Chu, Hua Yang and Gladys Chow. A Cooperative Database System (CoBase) for Query Relaxation in Proceedings of the Third International Conference on Artificial Intelligence Planning Systems. Edinburgh, May 1996.
[6] Dongwon Lee. "Query Relaxation for XML Model"
In Ph.D Dissertation, University of California, Los Angeles, June 2002
[7] M. Mitra, A. Singhal, and C. Buckley. "Improving Automatic Query Expansion". In ACM SIGIR, Melbourne, Autstralia, Aug 1998.

[8] W. W. Chu, Q. Chen, and A. Huang. "Query Answering via Cooperative Data Inference". J. Intelligent Information Systems (JIIS), 3(1):57–87, Feb. 1994.

[9] S. Chaudhuri and L. Gravano. "Evaluating Top-k Selection Queries".
In VLDB, Edinburgh, Scotland, Sep. 1999.

[10] World Wide Web Consortium. XML Path Language (XPath) Version 1.0. W3C Recommendation, Nov. 16, 1999. See http://www.w3.org/TR/xpath.html

[11] J. Robie, J. Lapp, D. Schach. XML Query Language (XQL). See http://www.w3.org/TandS/QL/QL98/pp/xql.html.

[12] International Organization for Standardization (ISO). Information Technology-Database Language SQL. Standard No. ISO/IEC 9075:1999. (Available from American National Standards Institute, New York, NY 10036, (212) 642-4900.)

[13] Java Compiler Compiler [tm] (JavaCC [tm]) - The Java Parser Generator. See https://javacc.dev.java.net/.

[14] JavaCC Grammar Repository. See http://www.cobase.cs.ucla.edu/pub/javacc/.

[15] Don Chamberlin, Jonathan Robie, and Daniela Florescu. Quilt: an XML Query Language for Heterogeneous Data Sources. In Lecture Notes in Computer Science, Springer-Verlag, Dec. 2000.

Appendix I: RLXQuery Terminals

UseXTah	::=	“USE-XTAH”
UseTah	::=	“USE-TAH”
NonRelaxable	::=	“!”
Tlide	::=	~
PoundSign	::=	“#”
SimilarTo	::=	“SIMILAR-TO”
BasedOn	::=	“BASED-ON”
PreferTag	::=	“PREFER-TAG”
RejectTag	::=	“REJECT-TAG”
Prefer	::=	“PREFER”
Reject	::=	“REJECT”
LabelV	:=	“LABEL-V”
LabelS	:=	“LABEL-S”
RelaxLevelS	:=	“RELAX-LEVEL-S”
RelaxLevelV	:=	“RELAX-LEVEL-V”
RelaxOrder	::=	“RELAX-ORDER”
RankBy	::=	“RANK-BY”
AtLeast	::=	“AT-LEAST”
Method	:=	“METHOD”

Appendix II: RLXQuery Non-Terminals

RLXQuery	::=	(FunctionDecl)* RLXQuerySpecification
RLXQuerySpecification	::=	RLXQuerySpecItem (("union" | "intersect" | "except")RLXQuerySpecItem)*
RLXQuerySpecItem	::=	RLXFLWORExpr | "(" RLXQuerySpecification ")"
RLXFLWORExpr	::=	(RLXForClause | RLXLetClause)+ (RLXWhereClause)? (OrderByClause)? ReturnClause (RelaxationOrderClause | AtLeastClause | RankMethodClause)*
RLXForClause	::=	<ForVariable> <VarName> <In> BExpr (<Comma> <VariableIndicator> <VarName> <In> BExpr)*
BExpr	::=	FLWORExpr | (SpecialPathExpr PreferTagClause? RejectTagClause?)
PreferTagClause	::=	"PREFER-TAG" <Lpar> GeneralStringList<Rpar>
RejectTagClause	::=	"REJECT-TAG" <Lpar> GeneralStringList <Rpar>
RLXLetClause	::=	<LetVariable> <VarName> ":=" BExpr (<Comma> <VariableIndicator> <VarName> ":="BExpr)*
RLXWhereClause	::=	"where" RLXWhereExpr
RLXWhereExpr	::=	RLXAndExpr ("or" RLXAndExpr)*
RLXAndExpr	::=	RLXCompExpr ("and" RLXCompExpr)*
RLXCompExpr	::=	CooperativeSearchCondition | ComparisionExpr
CooperativeSearchCondition	::=	CooperativeBooleanTerm ("or" CooperativeBooleanTerm)*
CooperativeBooleanTerm	::=	CooperativeBooleanFactor ("and" CooperativeBooleanFactor)*
CooperativeBooleanFactor	::=	CooperativeBooleanPrimary ("not" CooperativeBooleanPrimary)*
CooperativeBooleanPrimary	::=	CooperativePredicate (TahAliasLevelClause)? | CooperativePredicate (XTahAliasLevelClause)? | <Lpar> CooperativeSearchCondition <Rpar> (TahAliasLevelClause)? | <Lpar> CooperativeSearchCondition <Rpar> (XTahAliasLevelClause)?
TahAliasLevelClause	::=	UseTahClause ((VConditionLabel VRelaxationLevel?) | (VRelaxationLevel VConditionLabel?))? |VConditionLabel ((VRelaxationLevel UseTahClause?) | (UseTahClause

		VRelaxationLevel?))? |VRelaxationLevel ((VConditionLabel UseTahClause?) | (UseTahClause VConditionLabel?))?
UseTahClause	::=	<UseTah> StringLiteral
VConditionLabel	::=	<LabelV> StringLiteral
VRelaxationLevel	::=	<RelaxLevelV> NumericLiteral
XTahAliasLevelClause	::=	UseXTahClause ((SConditionLabel SRelaxationLevel?) | (SRelaxationLevel SConditionLabel?))? |SConditionLabel ((UseXTahClause SRelaxationLevel?) | (SRelaxationLevel | UseXTahClause)?)? |SRelaxationLevel (SConditionLabel UseXTahClause?) |(UseXTahClause SConditionLabel?))?
UseXTahClause	::=	`
SConditionLabel	::=	<LabelS> StringLiteral
SRelaxationLevel	::=	<RelaxLevelS> NumericLiteral
CooperativePredicate	::=	CooperativeCompPredicate | MultipleSimilarToPredicate
MultipleSimilarToPredicate	::=	PathExprList "SIMILARTO" GeneralValueList (BasedOnClause)?
PathExprList	::=	"(" PathExpr (",” PathExpr)* ")"
GeneralValueList	::=	<Lpar> GeneralStringList <Rpar> |<Lpar> GeneralNumericList <Rpar>
GeneralStringList	::=	StringLiteral (<Comma> StringLiteral)*
GeneralNumericList	::=	GeneralNumericElem (<Comma> GeneralNumericElem)*
GeneralNumericElem	::=	SimpleNumericExpr | NumericRange
NumericRange	::=	<Lbrack> NumericLiteral <Comma> NumericLiteral <Rbrack>
SimpleNumericExpr	::=	SimpleTerm ((<Plus>| <Minus>) SimpleTerm)*
SimpleTerm	::=	NumericLiteral ((<Multiply> | <Div>) NumericLiteral)*
BasedOnClause	::=	<Lpar> BasedOnPathExprList <Rpar>
BasedOnPathExprList	::=	PathExprWeightList (<Comma> PathExprWeightList)*
PathExprWeightList	::=	<Lpar> PathExprWeightElem <Rpar> (<Comma> <Lpar> PathExprWeightElem <Rpar>)*
PathExprWeightElem	::=	PathExpr <Comma> NumericLiteral

CooperativeCompPredicate	::=	CooperativePathExpr CompOpElem?
CompOpElem	::=	(GeneralComp (CooperativeValueElem| PathExpr) (ValueRejectionList?))| (SingleSimliarToElem BasedOnClause?)
GeneralComp	::=	"<" | ">" | "<=" | ">=" | "=" |"!="
CooperativePathExpr	::=	ApproximatePathExpr | NonRelaxablePathExpr | SpecialPathExpr
ApproximatePathExpr	::=	"~" "(" PathExpr ")"
NonRelaxablePathExpr	::=	"!" "(" PathExpr ")"
SpecialPathExpr	::=	<Root> SpecialRelativePathExpr? | <RootDescendants> SpecialRelativePathExpr | SpecialRelativePathExpr
SpecialRelativePathExpr	::=	SpecialStepExpr ((<NonRelaxable>)? (<Slash> | <SlashSlash>)) SpecialStepExpr)*
SpecialStepExpr	::=	SpecialAxisStep | SpecialFilterStep
SpecialAxisStep	::=	(SimpAbbrevForwardStep | AbbrevReverseStep) SpecialPredicates
SimpAbbrevForwardStep	::=	"@"? SimpNodeTest
AbbrevReverseStep	::=	"..."
SimpNodeTest	::=	"text" "(" ")"| QName | Wildcard
SpecialPredicates	::=	("[" SpecialPredicateExpr "]")*
SpecialPredicateExpr	::=	SpecialPredicateTerm ("and" SpecialPredicateTerm)*
SpecialPredicateTerm	::=	("contains(" PredicatePathExpr "," StringLiteral ")") | ("not (" "contains(" PredicatePathExpr "," StringLiteral "))") | (PredicatePathExpr CompOp PredicateValue)
PredicatePathExpr	::=	(("!")? ("/" | "//"))? PredicateRelativePathExpr
PredicateRelativePathExpr	::=	PredicateStepExpr (("!")? ("/" | "//") PredicateStepExpr)?
PredicateStepExpr	::=	AbbreForwardStep | Literal | "$" VarName
PredicateValue	::=	ApproximateValue | NonRelaxableValue | ConceptValue | Literal
SpecialFunctionCall	::=	"contains" "("SpecialPathExpr, StringLiteral")" | "count" "(" PathExpr ")" | "document" "(" StringLiteral ")"
SpecialFilterStep	::=	SpecialPrimaryExpr SpecialPredicates
SpecialPrimaryExpr	::=	NonRelaxableNode | StringLiteral | ("$" VarName) | SpecialFunctionCall | "."

NonRelaxableNode	::=	<NonRelaxable> Literal
SingleSimilarToElem	::=	<SimilarTo> ExactValueExpr
CooperativeValueElem	::=	ConceptualValue | ApproximateValue | NonRelaxableValue | ((ExactValueExpr | ValuePreferenceList) ValueRejectionList?)
ConceptualValue	::=	"#" StringLiteral
ApproximateValue	::=	"~" ExactValueExpr
NonRelaxableValue	::=	"!" ExactValueExpr
ValuePreferenceList	::=	<Prefer> GeneralValueList
ValueRejectionList	::=	<Reject> GeneralValueList
ExactValueExpr	::=	StringLiteral | GeneralNumericElem
AtLeastClause	::=	<At-Least> IntegerLiteral
RankMethodClause	::=	<Rank-By><Lbrack>RankItem (<Comma> RankItem)*<Rbrack> (<Method> StringLiteral)?
RankItem	::=	<Lpar> (StringLiteral (<Comma> NumericLiteral)?) <Rpar>
RelaxationOrderClause	::=	<RelaxOrder> RelaxOrderList
RelaxOrderList	::=	<Lpar> RelaxOrderElem <Rpar> | <Lbrace> RelaxOrderElem <Rbrace> | <Lbrack> RelaxOrderElem <Rbrack>
RelaxOrderElem	::=	StringLiterall (<Comma> (StringLiteral | RelaxOrderList))* | RelaxOrderList (<Comma> StringLiteral)+

Appendix III: Parser Tree Example #1

Query Example:

```
for $d in document("dblp.xml")/dblp
let $b := $d/paper PREFER-TAG("article", "proceedings")
where $b/tile = PREFER("XML", "semi-structured data") and $b/year >!2002
return $b
```

Parser's JJTree Output:

```
|XPath2
|   RLXQuery
|      RLXQuerySpecification
|         RLXQuerySpecItem
|            RLXFLWORExpr
|               RLXForClause
|                  VarName b
|                  In in
|                  BExpr
|                     SpecialPathExpr
|                        SpecialStepExpr
|                           SpecialFilterStep
|                              SpecialFunctionCall
|                                 DocumentsLpar document(
|                                 StringLiteral "bib.xml"
|                              SpecialPredicates
|                        SlashSlash //
|                        SpecialStepExpr
|                           SpecialAxisStep
|                              SimpAbbrevForwardStep
|                                 SimpNodeTest
|                                    QName book
|                              SpecialPredicates
|               RLXWhereClause
|                  Where where
|                  RLXWhereExpr
|                     RLXAndExpr
|                        RLXCompExpr
|                           CooperativeBooleanTerm and
|                              CooperativeBooleanTerm and
|                                 TahAliasLevelExpr
|                                    XTahAliasLevelExpr
|                                       SimilarToExpr
|                                          CooperativeCompPredicate
|                                             CooperativePathExpr
|                                                SpecialPathExpr
|                                                   SpecialStepExpr
|                                                      SpecialFilterStep
|                                                         ApproximateNode ~$
|                                                         VarName b
|                                                         SpecialPredicates
|                                                   Slash /
|                                                   SpecialStepExpr
|                                                      SpecialAxisStep
|                                                         SimpAbbrevForwardStep
|                                                            SimpNodeTest
|                                                               QName title
|                                                         SpecialPredicates
|                                                            Lbrack [
|                                                            SpecialPredicateExpr
|                                                               SpecialPredicateTerm
```

```
|                                         ContainsLpar contains(
|                                         PredicatePathExpr
|                                           PredicateStepExpr
|                                             Dot .
|                                         StringLiteral "XML"
|                                   Rbrack ]
|                           XTahAliasLevelClause
|                             UseXTahClause
|                               UseXTah USE-XTAH
|                               QName t1
|                         TahAliasLevelExpr
|                           XTahAliasLevelExpr
|                             SimilarToExpr
|                               CooperativeCompPredicate
|                                 CooperativePathExpr
|                                   SpecialPathExpr
|                                     SpecialStepExpr
|                                       SpecialFilterStep
|                                         VarName b
|                                         SpecialPredicates
|                                     SlashSlash //
|                                     SpecialStepExpr
|                                       SpecialAxisStep
|                                         SimpAbbrevForwardStep
|                                           SimpNodeTest
|                                             QName year
|                                         SpecialPredicates
|                                 CompOpElem
|                                   SpecialComp
|                                     Gt >
|                                   CooperativeValueElem
|                                     GeneralNumericElem
|                                       SimpleNumericExpr
|                                         SimpleTerm
|                                           IntegerLiteral 2001
|                           TahAliasLevelClause
|                             UseTahClause
|                               UseTah USE-TAH
|                               QName t2
|                       TahAliasLevelExpr
|                         XTahAliasLevelExpr
|                           SimilarToExpr
|                             CooperativeCompPredicate
|                               CooperativePathExpr
|                                 SpecialPathExpr
|                                   SpecialStepExpr
|                                     SpecialFilterStep
|                                       VarName b
|                                       SpecialPredicates
|                                   Slash /
|                                   SpecialStepExpr
|                                     SpecialAxisStep
|                                       SimpAbbrevForwardStep
|                                         SimpNodeTest
|                                           QName price
|                                       SpecialPredicates
|                               CompOpElem
|                                 SpecialComp
|                                   Lt <
|                                 CooperativeValueElem
|                                   NonRelaxableValue
|                                     id !
|                                     GeneralNumericElem
|                                       SimpleNumericExpr
|                                         SimpleTerm
|                                           IntegerLiteral 50
```

```
|                         TahAliasLevelClause
|                           VConditionLabel
|                             id LABEL-V
|                             QName t3
|           Return return
|           ExprSingle
|             UnaryExpr
|               PathExpr
|                 StepExpr
|                   FilterStep
|                     VarName b
|                     Predicates
|           AtLeastClause
|             id AT-LEAST
|             IntegerLiteral 5
|           RelaxationOrderClause
|             id RELAX-ORDER
|             RelaxOrderList
|               Lbrack [
|               RelaxOrderElem
|                 RelaxOrderList
|                   RelaxOrderElem
|                     QName t1
|                     QName t2
|                 QName t3
|               Rbrack ]
|           RankMethodClause
|             id RANK-BY
|             Lbrack [
|             RankItem
|               QName t2
|               DecimalLiteral 0.4
|             RankItem
|               QName t3
|               DecimalLiteral 0.5
|             RankItem
|               QName t1
|               DecimalLiteral 0.6
|             Rbrack ]
```

Appendix IV: Parser Tree Example #2

Query Example:

```
for $b in document("bib.xml")//book
where ~$b/title[contains(., "XML")] USE-XTAH t1 and
$b//year > 2001 USE-TAH t2 and $b/price < !50 LABEL-V t3
return $b
AT-LEAST 5
```

Parser's JJTree Output:

```
|XPath2
|   RLXQuery
|      RLXQuerySpecification
|         RLXQuerySpecItem
|            RLXFLWORExpr
|               RLXForClause
|                  VarName d
|                  In in
|                  BExpr
|                     SpecialPathExpr
|                        SpecialStepExpr
|                           SpecialFilterStep
|                              SpecialFunctionCall
|                                 DocumentsLpar document(
|                                 StringLiteral "dblp.xml"
|                              SpecialPredicates
|                        Slash /
|                        SpecialStepExpr
|                           SpecialAxisStep
|                              SimpAbbrevForwardStep
|                                 SimpNodeTest
|                                    QName dblp
|                              SpecialPredicates
|               RLXLetClause

|                  LetVariable let $
|                  VarName b
|                  ColonEquals :=
|                  BExpr
|                     SpecialPathExpr
|                        SpecialStepExpr
|                           SpecialFilterStep
|                              VarName d
|                              SpecialPredicates
|                        Slash /
|                        SpecialStepExpr
|                           SpecialAxisStep
|                              SimpAbbrevForwardStep
|                                 SimpNodeTest
|                                    QName paper
|                              SpecialPredicates
|                     PreferTagClause
|                        string PREFER-TAG(
|                        GeneralStringList
|                           StringLiteral "article"
|                           StringLiteral "proceedings"
|               RLXWhereClause
|                  Where where
|                  RLXWhereExpr
|                     RLXAndExpr
|                        RLXCompExpr
```

```
|                     CooperativeBooleanTerm and
|                       TahAliasLevelExpr
|                         XTahAliasLevelExpr
|                           SimilarToExpr
|                             CooperativeCompPredicate
|                               CooperativePathExpr
|                                 SpecialPathExpr
|                                   SpecialStepExpr
|                                     SpecialFilterStep
|                                       VarName b
|                                       SpecialPredicates
|                                   Slash /
|                                   SpecialStepExpr
|                                     SpecialAxisStep
|                                       SimpAbbrevForwardStep
|                                         SimpNodeTest
|                                           QName tile
|                                       SpecialPredicates
|                               CompOpElem
|                                 SpecialComp
|                                   Equals =
|                                 CooperativeValueElem
|                                   ValuePreferenceList
|                                     string PREFER(
|                                     GeneralValueList
|                                       GeneralStringList
|                                         StringLiteral "XML"
|                                         StringLiteral "semi-structured data"
|                       TahAliasLevelExpr
|                         XTahAliasLevelExpr\
|                           SimilarToExpr
|                             CooperativeCompPredicate
|                               CooperativePathExpr
|                                 SpecialPathExpr
|                                   SpecialStepExpr
|                                     SpecialFilterStep
|                                       VarName b
|                                       SpecialPredicates
|                                   Slash /

|                                   SpecialStepExpr
|                                     SpecialAxisStep
|                                       SimpAbbrevForwardStep
|                                         SimpNodeTest
|                                           QName year
|                                       SpecialPredicates
|                               CompOpElem
|                                 SpecialComp
|                                   Gt >
|                                 CooperativeValueElem
|                                   NonRelaxableValue
|                                     id !
|                                     GeneralNumericElem
|                                       SimpleNumericExpr
|                                         SimpleTerm
|                                           IntegerLiteral 2002
|             Return return
|             ExprSingle
|               UnaryExpr
|                 PathExpr
|                   StepExpr
|                     FilterStep
|                       VarName b
|                       Predicates
```