Appendix A

List of Abbreviations

API	Application Programming Interface
C14N	Canonicalization
DAWG	Data Access Working Group [Pru06]
DSA	Digital Signature Algorithm [FIP95b]
EBNF	Extended Backus-Naur Form [ISO96]
FOAF	Friend of a Friend [BM04]
ICRA	Internet Content Rating Association
ID	Identifier
ISIN	International Securities Identifying Number [ISO01]
ISO	International Organization for Standardization
MD5	Message-Digest Algorithm 5 [Riv92]
NG4J	Named Graph API for Jena [BC06]
N3	Notation 3 [BL98]
OWL	Web Ontology Language [MvH04]
PDF	Portable Document Format [MvH04]
PICS	Platform for Internet Content Selection [MKRT96]
RDF	Resource Description Framework [Bec04b]
RFC	Request for Comments
RSA	Rivest, Shamir and Adleman [KS98]
RSS	Really Simple Syndication [RSS05]
SPARQL	Simple Protocol and RDF Query Language [PS05]
SWP	Semantic Web Publishing
TriG	Triple Graph Syntax [Biz05]
TriX	RDF Triples in XML Syntax [CS04a]
URI	Uniform Resource Identifier [BLFM98]
URL	Uniform Resource Locator [BLFM98]
URN	Uniform Resource Name [BLFM98]
UUID	Universally Unique Identifier [LMS05]
W3C	World Wide Web Consortium
WIQA-PL	Web Information Quality Assessment Policy Language
XHTML	Extensible HyperText Markup Language [Ste02]
XML	Extensible Markup Language [BPSMM00]
Xpath	XML Path Language [CD99]
XSLT	XSL Transformation [Cla99]

Appendix B

Grammar of the TriG Syntax

This appendix contains the Extended Backus-Naur Form (EBNF) [ISO96] grammar definition of the TriG syntax described in Section 5.3.2. A TriG document is a Unicode [The04] character string in the language defined by the following grammar, starting with the TriGDoc production.

```
1.  TriGDoc    ::= statement*
2.  statement  ::= directive ws* '.' ws* | graph ws* | comment | ws+
3.  directive  ::= '@prefix' ws+ prefixID ws+ uriref
4.  graph      ::= graphName? ws* ':-'? ws* '{' ( ws* (( triples '.' ) |
5.                 comment ))* ws* triples? ws* '}'
6.  graphName  ::= resource
7.  triples    ::= subject ws+ predicateObjectList
8.  predicateObjectList ::= verb ws+ objectList ( ws+ ';' ws* verb
9.                 ws+ objectList)*
10. objectList ::= object (ws+ ',' ws* object)*
11. verb       ::= predicate | a
12. comment    ::= '#' ( character - ( #xD | #xA ) )*
13. subject    ::= resource | blank
14. predicate  ::= resource
15. object     ::= resource | blank | literal
16. literal    ::= langString | datatypeString | integer
17. langString ::= '"' string '"' ( '@' language )?
18. datatypeString ::= '"' string '"^^' (uriref | qname)
19. integer    ::= [0-9]+
20. blank      ::= nodeID | '[]' | '[' ws* predicateObjectList ws* ']' |
21.                collection
22. itemList   ::= object (ws+ object)*
23. collection ::= '(' itemList? ')'
24. resource   ::= uriref | qname
25. nodeID     ::= '_:' name
26. qname      ::= name? ':' name?
27. prefixID   ::= ':' | name ':'
28. uriref     ::= '<' IRI '>' # where IRI matches RFC 3987
29. language   ::= [a-z]+ ('-' [a-z0-9]+ )*
30. name       ::= [A-Za-z][A-Za-z0-9_]*
31. a          ::= 'a' # where 'a' is equivalent to the qname rdf:type
32. string     ::= character*
33. ws         ::= #x9 | #xA | #xD | #x20
34. character  ::= Unicode character in the range U+0 to U+10FFFF
```

Figure B.1: EBNF grammar definition of the TriG syntax.

Appendix C

Grammar of the WIQA-PL Policy Language

This appendix contains the Extended Backus-Naur Form (EBNF) [ISO96] grammar definition of the WIQA-PL policy language described in Chapters 9 and 10. A WIQA-PL policy suite is a Unicode [The04] character string in the language defined by the following grammar, starting with the PolicySuite production.

```
1.  PolicySuite          ::= PrefixDeclaration*
2.                           Policy+
3.  PrefixDeclaration    ::= 'PREFIX' PrefixID Uriref
4.  Policy               ::= PolicyName
5.                           PolicyDescription?
6.                           PolicyPattern
7.                           RDFExplanationClause
8.  PolicyName           ::= 'NAME' Literal
9.  PolicyDescription    ::= 'DESCRIPTION' Literal
10. PolicyPattern        ::= 'PATTERN' PatternSet
11  PatternSet           ::= '{' ExplanationClause?
12.                          GraphPattern*
13.                          FilterClause* '}'
14. FilterClause         ::= 'FILTER' FilterExpression '.'
15. ExplanationClause    ::= 'EXPL' ExplanationTemplates '.'
16. GraphPattern         ::= GraphName '{'
17.                          ExplanationClause?
18.                          TriplePattern+
19.                          FilterClause* '}'
20. GraphName            ::= 'GRAPH' VariableOrUriOrANY
21. TriplePattern        ::= URIOrBnodeOrVariableOrReference
22.                          URIOrVariableOrReference
23.                          URIOrBnodeOrLiteralOrVariableOrReference '.'
24. VariableOrUriOrANY   ::= Variable | URI | 'ANY'
25. URIOrBnodeOrVariableOrReference
26.                      ::= URI | Bnode | Variable | Reference
27. URIOrVariableOrReference
28.                      ::= URI | Variable | Reference
29. URIOrBnodeOrLiteralOrVariableOrReference
30.                      ::= URI | Bnode | Literal | Variable |
31.                          Reference
32. Variable             ::= '?' String
33. Reference            ::= '?GRAPH' | '?SUBJ' |'?PRED' | '?OBJ'
34. FilterClause         ::= 'FILTER' Expression | FunctionCall
35. Expression           ::= '(' ConditionalOrExpression ')'
36. ConditionalOrExpression ::= ConditionalAndExpression ( '||'
36.                          ConditionalAndExpression )*
37. ConditionalAndExpression ::= LogicalValue ( '&&' LogicalValue )*
38. LogicalValue       ::= RelationalExpression
39. RelationalExpression ::= NumericExpression ( '=' NumericExpression |
40.                          '!=' NumericExpression |
41.                          '<'  NumericExpression  |
42.                          '>'  NumericExpression |
43.                          '<=' NumericExpression |
44.                          '>=' NumericExpression )?
45. NumericExpression    ::= AdditiveExpression
```

Figure C.1: EBNF grammar of the WIQA-PL policy language - Part 1.

```
46. AdditiveExpression   ::= MultiplicativeExpression (
47.                          '+' MultiplicativeExpression |
48.                          '-' MultiplicativeExpression )*
49. MultiplicativeExpression ::= UnaryExpression ( '*' UnaryExpression |
50.                          '/' UnaryExpression )*
51. UnaryExpression      ::= '!' PrimaryExpression |
52.                          '+' PrimaryExpression |
53.                          '-' PrimaryExpression |
54.                          PrimaryExpression
55. PrimaryExpression    ::= Expression | FunctionCall | URIref |
56.                          RDFLiteral | NumericLiteral | BooleanLiteral |
57.                          BlankNode | Variable
58. FunctionCall         ::= RDFrelatedFunction | CastingOrExtensionFunction
59. CastingOrExtensionFunction ::=  URIref ArgList
60. ArgList              ::= ( '(' NIL | Expression ( ',' Expression )* ')' )
61. RDFrelatedFunction   ::= 'str' '(' Expression ')' |
62.                          'lang' '(' Expression ')' |
63.                          'datatype' '(' Expression ')' |
64.                          'isUri' '(' Expression ')' |
65.                          'isBlank' '(' Expression ')' |
66.                          'isLiteral' '(' Expression ')' |
67.                          'regex' '(' Expression ',' Expression (
68.                          ',' Expression )? ')'
69. ExplanationClause    ::= 'EXPL' ExplanationTemplate '.'
70. ExplanationTemplate  ::= ( Literal | Variable | ExtensionFunctionURI )+
71. RDFExplanationClause ::= 'CONSTRUCT' 'EXPLANATION' ConstructTemplate
72. ConstructTemplate    ::= '{' ConstructPattern+ '}'
73. ConstructPattern     ::= URIOrBnodeOrVariableOrReference
74.                          URIOrBnodeOrVariableOrReference
75.                          URIOrBnodeOrLiteralOrVariableOrReference '.'
76. URI                  ::= Uriref | Qname
77. Qname                ::= Name? ':' Name?
78. Uriref               ::= '<' IRI '>' # where IRI matches RFC 3987
79. Bnode                ::= NodeID | '[]'
80. NodeID               ::= '_:' name
81. Literal              ::= LangString | DatatypeString | Integer
82. LangString           ::= '"' String '"' ( '@' Language )?
83. Language             ::= [a-z]+ ('-' [a-z0-9]+ )*
84. DatatypeString       ::= '"' String '"^^' (URI)
85. Integer              ::= [0-9]+
86. Variable             ::= '?' Name
87. PrefixID             ::= ':' | Name ':'
88. Name                 ::= [A-Za-z][A-Za-z0-9_]*
89. String               ::= Character*
90. Character            ::= Unicode character in the range U+0 to
                             U+10FFFF
```

Figure C.2: EBNF grammar of the WIQA-PL policy language - Part 2.

Appendix D

German Summary

Web-basierte Informationssysteme, wie Suchmaschinen, Nachrichtenportale, Finanzportale, elektronische Märkte oder virtuelle Gemeinschaften, ermöglichen den Zugriff auf Informationen aus einer Vielzahl von Quellen. Die Qualität der angebotenen Informationen ist sehr unterschiedlich, da die Informationsanbieter verschiedene Wissensstände, verschiedene Meinungen und unterschiedliche Ziele haben. In Alltagssituationen wenden wir intuitiv eine Vielzahl unterschiedlicher Verfahren zur Auswahl qualitativ hochwertiger Informationen an. Diese Auswahlverfahren stützen sich auf den Inhalt der Informationen, auf Metainformationen über die zur Auswahl stehenden Informationen, auf Hintergrundinformationen über die Informationsanbieter sowie auf Empfehlungen Dritter. Welches Verfahren wir anwenden hängt vom jeweiligen Kontext, den zur Verfügung stehenden Qualitätsindikatoren sowie unseren subjektiven Präferenzen ab.

In der vorliegenden Arbeit wird ein Filter-Framework entwickelt, das es den Nutzern web-basierter Informationssysteme ermöglicht, ein ähnlich breites Spektrum unterschiedlicher Verfahren zur Auswahl qualitativ hochwertiger Informationen anzuwenden. Das Framework besteht aus zwei Artefakten: Einem Datenmodell zur integrierten Repräsentation von Informationen und qualitätsbezogenen Metainformationen sowie einer formalen Sprache zur Formulierung von Filterpolitiken. Das entwickelte Framework wird in einen Webbrowser integriert und im Rahmen eines Investment-Szenarios evaluiert. Die Arbeit gliedert sich in drei Teile:

1. Informationsqualität und das Web

Der erste Teil der Arbeit gibt einen Überblick über die bisherige wissenschaftliche Arbeit zum Thema Informationsqualität im Kontext web-basierter Informationssysteme. Informationsqualität wird in der Literatur

überwiegend als die subjektive Nützlichkeit von Informationen aufgefasst. Die subjektive Nützlichkeit wird durch Qualitätskriterien, wie Korrektheit, Verständlichkeit, Aktualität und Vollständigkeit, bestimmt. Die Arbeit gibt einen Überblick über verschiedene Metriken zur Quantifizierung der Kriterien und diskutiert die Anwendbarkeit dieser Metriken im Rahmen web-basierter Systeme. Die Metriken werden in die Gruppen inhalts-bezogene, kontext-bezogene und bewertungs-bezogene Metriken klassifiziert. Anschließend wird das Konzept der Qualitäts-Filter-Politik eingeführt und verschiedene Politiken diskutiert, die ein Investor zur Auswahl qualitativ hochwertiger Finanzinformationen einsetzen kann.

2. Repräsentation von Metainformationen

Die Einschätzung der Qualität von Informationen stützt sich auf unterschiedliche Qualitätsindikatoren, wie beispielsweise Hintergrundinformationen oder Bewertungen. Um verschiedene Politiken zur Auswahl qualitativ hochwertiger Informationen anwenden zu können, ist es daher erforderlich, Informationen zusammen mit qualitätsbezogenen Metainformationen in einem integrierten Datenmodell zu repräsentieren. Der zweite Teil der Arbeit analysiert inwieweit sich das Resource Description Framework (RDF), ein aktuelles Datenmodell für web-basierte Informationssysteme, zur gemeinsamen Repräsentation von Informationen und qualitätsbezogenen Metainformationen eignet. Es wird festgestellt, dass der RDF Reifikations-Mechanismus den Anforderungen nicht gerecht wird. Daher wird die Erweiterung des RDF Datenmodells zum Named Graphs Datenmodell vorgeschlagen. Anschließend wird anhand des Investment-Szenarios aufgezeigt, wie sich Named Graphs zur Repräsentation qualitätsbezogener Metainformationen einsetzen lassen. Das Named Graph Datenmodell wird bereits von mehreren Software-Toolkits implementiert und in mehreren Projekten praktisch eingesetzt. Das Datenmodell wurde von World Wide Web Consortium (W3C), dem führenden Standardisierungsgremium für Web-Technologien, in den SPARQL Standard übernommen.

3. Das WIQA Framework

Im dritten Teil der Arbeit wird eine formale Sprache zur Formulierung von Filterpolitiken entwickelt sowie deren Implementierung im Rahmen des WIQA Framework beschrieben. Die entwickelte Sprache ist an SPARQL, einer vom W3C standardisierten Abfragesprache für RDF Daten, angelehnt. Da unterschiedliche Anwendungsbereiche spezielle, anwendungsbereich-spezifische Metriken zur Quantifizierung von Qualitätskriterien erfordern,

verfügt die entwickelte Sprache über eine offene Schnittstelle zur Einbindung von Metriken. Die Nachvollziehbarkeit des Filterungsprozesses ist ein entscheidender Faktor, der das Vertrauen der Nutzer in die Qualität von positiv gefilterten Informationen bestimmt. Um diese Nachvollziehbarkeit zu gewährleisten, kann das WIQA-Famework natürlichsprachliche Erklärungen erzeugen, inwiefern Informationen einer gegebenen Politik entsprechen.

Das Named Graphs Datenmodell wurde in Form der NG4J - Named Graphs API for Jena implementiert. Die entwickelte Sprache zur Formulierung von Filterpolitiken wurde im Rahmen der WIQA Filtering and Explanation Engine implementiert. Beide Software-Komponenten wurden in einen Webbrowser integriert und es wird anhand des Investment-Szenarios aufgezeigt, wie ein Investor den Browser zur Auswahl qualitativ hochwertiger Finanzinformationen einsetzen kann.

Webseite zur Arbeit

Der komplette englischsprachige Text der Arbeit, der Quelltext der entwickelten Software-Komponenten sowie Beispieldaten zum Investment-Szenario befinden sich auf der Webseite zur Arbeit http://sites.wiwiss.fu-berlin.de/suhl/bizer/WIQA/.