Mathematical Services Query Language: Design, Formalization, and Implementation

Rebhi Baraka*
Research Institute for Symbolic Computation (RISC-Linz)
Johannes Kepler University, Linz, Austria
rbaraka@risc.uni-linz.ac.at

September 2005

Abstract

The Mathematical Services Query Language (MSQL) is a content-based query language developed for querying mathematical descriptions in the form of the Mathematical Services Description Language (MSDL). It provides the user with the ability to query the contents of MSDL documents published in the MathBroker registry. It complements the metadata-based querying facility of the MathBroker registry which facilitates querying of metadata accompanying descriptions published in the registry. In this report, we present the design, the formal definition, and the implementation of MSQL with use-cases and examples demonstrating its usage.

Contents

*This work was sponsored by the FWF Project P17643-NO4 "Mathbroker II: Brokering of Distributed Mathematical Services".

1 Introduction

Mathematical services are Web services presenting solutions to mathematical problems. In the MathBroker project [23], the Mathematical Services Description Language (MSDL) [5] [25] was developed to adequately describe mathematical services respectively their constituent entities such as problems, algorithms, implementations, and machines. To facilitate the process of publishing and discovering mathematical services, we developed the MathBroker registry [3] [4] where MSDL descriptions of services are published so clients can discover them through browsing the registry or by querying it.

The querying capabilities of the MathBroker registry are rather limited, they only allow to query metadata accompanying the service when published in the registry. Although metadata such as names, unique identifiers, classifications, and associations of the service are useful in some cases, e.g. when we know the service we want to use, or when we are seeking a service based on its associations to other services, they are insufficient in many cases as a basis for service discovery. Therefore it is necessary to resort to the contents of the MSDL description of a service where sufficient and complete information can be found. So to overcome this limitation of the current search facilities of the registry, we designed and implemented the Mathematical Services Query Language (MSQL) for retrieving and querying the contents of MSDL documents that are published in the MathBroker registry.

MSQL has a set of features compatible with today's XML query languages to make the process of querying, the XML-based, MSDL documents easy and efficient. Our aim is that we want a simple, small, yet general querying tool for querying MSDL that can be extended later to deal with the semantically rich content of MSDL and/or query other XML content. This lead us to start designing the language taking into consideration specific characteristics of current XML query languages. From XPath [11], QUILT [6], and XQuery [12] MSQL is influenced by the syntax for navigating in the hierarchical structure of MSDL documents. From SQL [1] MSQL is influenced by the idea of a series of clauses based on keywords that provide a query pattern (the SELECT-FROM-WHERE pattern in SQL).

The rest of this report deals with the design, formalization, and implementation of MSQL. Section 2 describes the MathBroker framework, on which this work depends and builds upon, starting with the information model of MSDL and the current approach used to search MathBroker registry for MSDL documents. Section 3 presents the design of the language stating its design goals, its structure, and some use cases. Section 4 describes the abstract language and defines its formal semantics. Section 5 presents the implementation of MSQL describing the overall architecture, the strategy, some implementation details, and the API of MSQL and its usage. Section 6 briefly describes some of the works related to MSQL. Finally Section 7 concludes the report with the future direction of our work.

2 The MathBroker Framework

2.1 The MSDL Information Model

Among the requirements of Web services is the need to advertise them by providers, e.g., to a Web registry, such that they can be discovered by clients; therefore they need to be described in a machine-understandable format. In the case of mathematical Web services, these descriptions must be based on formal mathematics. Figure 1 illustrates the MathBroker information model for the description of mathematical Web services. It shows the kinds of entities that can be associated to a service and the relationships among them.

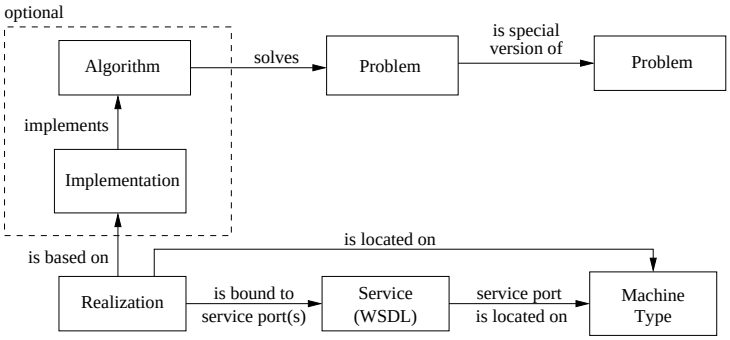

Figure 1: The MathBroker Information Model

The information model is implemented as a highly structured language called Mathematical Services Description Language (MSDL) [5, 25]. MSDL was developed in the frame of MathBroker project [23] with influences from the MONET project [24].

The entities of the model are as follows:

Problem A problem can be specified by input parameters, an input condition, output parameters, and an output condition. It can be a special version of another problem.

Algorithm An algorithm is described by (a link to the description of) the problem it solves, as well as by time and memory complexity, termination conditions, and bibliographical information. Bibliographical information may be given by Dublin Core Metadata.

Implementation An implementation is described by referring to the algorithm on which it is based (or optionally the problem it solves), the software, programming language, and numerical libraries that are used, and optionally time and memory efficiency w.r.t. some reference architecture.

Realization A realization of a service is the description that brings together the abstract specification of the service functionality with the actual details of the interface. Hence, it contains both a reference to either the underlying software implementation, or to the algorithm or the problem, and to a WSDL description of the service interface. It may also specify additional information on the hardware on which the service is running.

Machine A machine that a service runs on can be described by its processor type and speed, by its memory size, and by the type of the operating system it uses.

A skeleton of a service description in MSDL is shown below. Appendix C contains a complete description of such a service.

```
<monet:definitions
  ...
  <mathb:machine_hardware name="perseus.risc.uni-linz.ac.at">
    ...
  </mathb:machine_hardware>

  <monet:problem name="indefinite-integration">
    ...
  </monet:problem>

  <monet:algorithm name="RischAlg">
    ...
  </monet:algorithm>

  <monet:implementation name="RImpl">
    ...
  </monet:implementation>

  <monet:service name="RRISC">
      ...
  </monet:service>
</monet:definitions>
```

In the conventional Web services architecture, a registry is a software application for publishing and discovering information about Web services. Following this convention, we designed and implemented the MathBroker registry [3, 4] which provides a set of functionalities to facilitate the process of publishing and discovering mathematical services. We based it on the ebXML registry specification [18] and reference implementation [16] which has a generic and extensible information model [17] that we extended to accommodate the MathBroker information model (See Figure 1) as shown in Figure 2.

The MathBroker registry implementation [28] incorporating this model allows MSDL descriptions to be processed, classified according to its basic entities, have associations among them, classified according to some classification taxonomies, published in the registry, and queried.

2.2 Querying MSDL Descriptions

The way a service, respectively its constituent MSDL entities, is published in the MathBroker registry influences the way it can be discovered. An entity is published as a registry object that

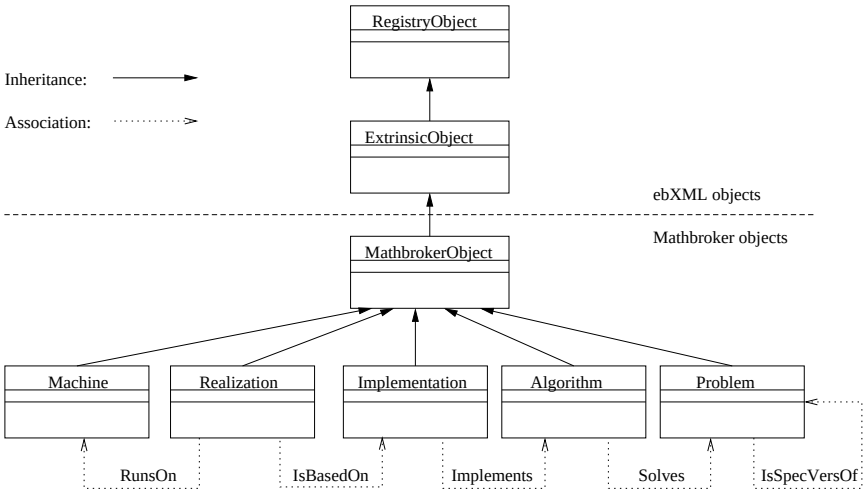

Figure 2: The MathBroker Information Model Registry-Inheritance Hierarchy

consists of:

- minimal metadata such as name, unique identifier, classifications of the objects against classification schemes, and associations of each object to other objects, and
- an MSDL description that is stored as a repository item in the registry file system (called repository).

Querying in this case can be performed at two levels:

- **Registry-level (metadata-based) querying:** this is supported by the registry's underlying query capabilities such as querying registry objects by name, by unique identifier, or by classification depending on the metadata of the objects' submitted to the registry.
- **MSDL-level (content-based) querying:** this involves MSDL descriptions of published registry objects. These descriptions are stored in the registry as repository items when their objects are published to the registry. This level depends on the first level because in order to access the stored description we need to perform a registry-level querying for the objects owning these descriptions, e.g., querying the registry for descriptions of a given object (entity) type classified under a given classification concept.

The second level is not supported by the registry and our goal is to develop a query language that facilitates this type of querying on top of the first level. We next set the requirements for the language for querying MSDL descriptions. We take into account the MSDL representation which is basically a schema-based XML document and the environment, i.e., the registry, where such descriptions are stored.

3 Design of MSQL

Based on the need to query the contents of MSDL documents published in the MathBroker registry, in this section we develop the language building blocks stating its design goals (the requirements), its constructs (by explanation and use cases), and its concrete grammar.

3.1 Motivation

Given a set of MSDL documents (such as the one in Appendix C) published in the registry, we would like to perform the following:

> find all problems under classification concept "/GAMS/Symbolic Computation" whose first input argument has type "integer".

To accomplish this request, we would do the following:

- consider the classification "/GAMS/Symbolic Computation" and fetch each document with entity type "problem" beneath it;
- process each document and return it if it satisfies the following criteria: *the first "input" argument occurring in the "problem" node is of type "integer".*

The returned documents may need to be sorted before being returned as a result.

Performing the first step involves contacting the registry and fetching the candidate documents from it. Performing the second step involves processing the returned candidate documents to see if they satisfy the stated criteria. It also involves formulating the criteria in a format that is suitable for the contents of the MSDL documents. Our objective is develop a query language to accomplish these steps. We next state them as requirements for the language and formulate its syntax based on these requirements.

3.2 Requirements of MSQL

We state a minimal set of characteristics for a mathematical query language that stems from its anticipated use as a query language for MSDL documents published in the MathBroker registry. These requirements are the following:

- **Precise semantics:** The language should have a formal semantics to make the intended meaning unambiguous. This semantics should also provide a straight forward reference for evaluating the correctness of the implementation.
- **Functional:** The language should specify what is to be done; how is it done is left to the implementation. It should support different kinds of expressions and functions.
- **Registry interfacing:** The language should have the functionality to access the registry in order to determine and retrieve the candidate collection of MSDL documents.
- **Composing a query:** The language queries should be composed in a concise human-readable query syntax.
- **Query operations:** The operations that have to be supported by MSQL are:
  - **Retrieval:** Retrieving an MSDL document based on the type of entity it describes and on the registry classification concepts under which this entity is classified.
  - **Selection:** Choosing an MSDL document from the candidate documents based on content, structure, or attributes.
  - **Evaluation:** Ultimately selecting a document based on the evaluation of a semantic condition expressed by a logical formula.
- **Input and output:** The input to a query should be MSDL documents satisfying the "Retrieval" requirement. The output of a query are MSDL documents.

- **Result views:** The language should support ordered and unordered views of query results.

It is not required that the language is capable of restructuring a document or returning portions of a document as a result. This is not a disadvantage because our goal is to return whole documents that can be further processed by other applications.

Based on these requirements we have designed the language described in the following subsection.

3.3 The Data Model

As stated in the requirements, the input and output to an MSQL query are MSDL documents. In section 2.1 we presented the MSDL information model and showed a skeleton of a description in MSDL (Appendix C shows a complete service description.) An MSDL description is basically a schema-based XML representation stored in the registry as an XML document. This document is modeled as a tree of nodes where each node may have a sequence of child nodes. A child node can be an element node, an attribute node, or a text node. Attribute and text nodes are always leaf nodes.

In the MSDL fragment below, `problem` is the root node, `OMOBJ` is a child node with `problem` as its parent (assuming the "..." represent siblings of `OMOBJ`). The `OMS` node (on line number 5) has an `OMA` node as its parent and has two attribute nodes as its children; `cd=‘‘arith1’’` and `name=‘‘plus’’`. The `OMI` element node (on line number 9) has the text node `2` as its child.

```
 1 <problem>
 2  ...
 3  <OMOBJ>
 4   <OMA>
 5     <OMS cd="arith1" name="plus"/>
 6     <OMA>
 7       <OMS cd="arith1" name="power"/>
 8       <OMV name="x"/>
 9       <OMI>2</OMI>
10     </OMA>
11     <OMV name="y"/>
12   </OMA>
13 </OMOBJ>
14 ...
15 </problem>
```

3.4 The Query Structure

The general structure of a query in MSQL is:

```
SELECT EVERY|SOME <entity>
FROM <classificationConcept>
WHERE <expression>
ORDERBY <expression> ASCENDING|DESCENDING
```

The query has four main clauses: the SELECT clause, the FROM clause, the WHERE clause, and the ORDERBY clause. The FROM clause and part of the SELECT clause, namely `entity`, are registry-oriented, i.e., their functionality is applied to the registry. This satisfies

the "Registry interfacing" requirement by determining the type of document, its classification in the registry, and retrieving it from the registry. This is a crucial issue of the language from the point of efficiency role since it limits the range of documents to be queried to those who are of type `entity` and classified under `classificationConcept`. The `entity` types as stated in the information model (see Figure 2) are PROBLEM, ALGORITHM, IMPLEMENTATION, REALIZATION, and MACHINE. `ClassificationConcept` is a node in a given classification taxonomy of the registry, e.g., "/GAMS/Symbolic Computation" in the GAMS classification of mathematical subjects. The SELECT clause also determines whether to return some or all of the resulting documents to the user by its SOME or EVERY clause.

The WHERE and the ORDERBY clauses apply their `expression` parts to each candidate document retrieved from the registry. The expression of the WHERE clause is a logical condition: if it is evaluated to true, the document is considered as (part of) the result of the query. The ORDERBY clause sorts the resulting documents in ASCENDING or DESCENDING order based on comparison criteria resulting from the evaluation of its expression on each document.

We illustrate the structure described so far by a concrete query use case.

Use Case 1. *Find all problems under the classification concept "/GAMS/Symbolic Computation" with first input having type integer and order them according to their names in descending order.*

```
SELECT EVERY problem
FROM /GAMS/Symbolic Computation
WHERE //body/input[1]/signature/om:OMOBJ/
       om:OMS[1][(( @name = "Z" )
        and ( @cd = "setname1" ))]
ORDERBY /problem/@name descending
```

This query asks for every "problem", i.e., every document of type "problem" classified under "/GAMS/Symbolic Computation" that satisfies the WHERE expression. The resulting documents are to be sorted in descending order according to their names. The core of the query is its WHERE expression which allows us to express the first input and check its type. The structure of such expressions is explained in the following subsection.

3.5 MSQL Expressions

Since we are designing a light-weight query language, we have specified a minimal set of expressions that are necessary to address the contents of the target MSDL documents. MSQL expressions include: path expressions that can access every part of an MSDL document; expressions involving logical, arithmetic, and comparative operators; conditional expressions; quantified expressions; functions; and variable bindings. They satisfy the "Functional" requirement of the language. The various kinds of expressions are described below.

Path Expressions

A path expression is basically a special XPath [11] expression which locates nodes within a document tree (see Section 3.3.) Path expressions transform transforming a sequence of nodes of a given document to another sequence of nodes by navigating through the given hierarchical structure.

A path expression in MSQL takes the following form:

```
<pathExpr>::= ((/ | //) <step> | . | ([<expression>])*)+
<step>::= <string> | @<string>
```

It consists of a series of one or more "steps", separated by "/" or "//", and optionally starting with "/" or "//". The result of each step is a set of nodes in the document.

A "/" at the start of a path expression denotes the root of the current document (sub)tree (i.e., the document root itself, if the path expressions occurs at the top-level of the query expression).

A "//" at the start of a path expression denotes every (descendent) node in a subtree of the current document root that is named as indicated by the step immediately following. This means that the entire root node tree hierarchy is to be navigated with every node in turn acting as the current node.

Every subsequent step in a path expression takes the set of nodes constructed from the predecessor step and constructs a new set of nodes.

We are now going to present some examples, based on the MSDL document shown below:

```
 1     <problem name="indefinite-integration">
 2     <header></header>
 3     <body>
 4       <input name="f">
 5         <signature>
 6           <OMOBJ>
 7             <OMA>
 8               <OMS cd="sts" name="mapsto"></OMS>
 9               <OMS cd="setname1" name="R"></OMS>
10               <OMS cd="setname1" name="R"></OMS>
11               <OMS cd="setname1" name="R"></OMS>
12             </OMA>
13           </OMOBJ>
14         </signature>
15       </input>
16       <output name="i">
17         <signature>
18           <OMOBJ>
19             <OMA>
20               <OMS cd="sts" name="mapsto"></OMS>
21               <OMS cd="setname1" name="R"></OMS>
22               <OMS cd="setname1" name="R"></OMS>
23               <OMS cd="setname1" name="R"></OMS>
24             </OMA>
25           </OMOBJ>
26         </signature>
27       </output>
28       <post-condition>
29         <OMOBJ>
30           <OMA>
31             <OMS cd="relation1" name="eq"></OMS>
32             <OMV name="i"></OMV>
33             <OMA>
34               <OMS cd="calculus1" name="indefint"></OMS>
35               <OMV name="f"></OMV>
36             </OMA>
```

```
37            </OMA>
38          </OMOBJ>
39        </post-condition>
40      </body>
41    </problem>
```

- The expression

  `/problem/body/input`

  selects the `input` node (i.e., the `input` node from lines 4 to 15) as follows: The first "/" establishes the root node (i.e., the document node) as the current node for the step immediately following. The result of this step is the `problem` node. Similarly, the "body" step selects the `body` node that is the child of the `problem` node. The same applies to the "input" step which results in selection of the `input` node which is the child of the `body` node.

- The expression

  `//input`

  returns the same result (the `input` node) as the previous example. It selects it as follows: the "//" establishes any (descendant) `input` child node of the root node for the step "input".

- The expression

  `//OMA`

  selects all `OMA` (descendent) nodes of the root node (i.e., the `OMA` node from lines 7 to 9), the `OMA` node from lines 30 to 37, and the `OMA` node from lines 33 to 36).

- The expression

  `//OMA/OMV`

  selects all `OMV` nodes that are children of an `OMA` node (the `OMV` node on line 32 and the `OMV` node on line 35).

A ".", i.e., "self", denotes the current node itself. For example:

- If the current node is problem, then the expression

  .

  selects the problem node.

Attributes are specified with a preceding `@` (e.g. `@String`). For example:

- The attribute in the expression

  `/problem/@name`

  selects the name attribute of the `problem` node.

The resulting document set may be filtered by a predicate which consists of an expression enclosed in square brackets `[ ]`. The predicate expression can either be logical which evaluates to a truth value or numeric which evaluates to a numeric value. The following examples illustrates both of them:

- The expression

  `/problem/body/input//OMA/OMS[@cd = ‘‘sts’’]`

  has the logical predicate `[@cd = ‘‘sts’’]`. It selects only the first (on line 8) `OMS` node with the `cd` attribute value “sts” among the four available `OMS` child nodes of node `OMA`.

- The expression

  `/problem/body/input//OMA/OMS[1]`

  has the numeric predicate `[1]`. It selects the first `OMS` node (on line 8) from the four available child nodes of the `OMA` node.

- The expression

  `/problem/body/input//OMA/OMS[3][(@name = ‘‘R’’) and (@cd =‘‘setname1’’)]`

  uses both a logical and a numeric predicate. The first predicate is numeric and it specifies the third `om:OMS` child node of the `OMA` node . The second one is logical and uses the `=` and the `and` operators to evaluate to a truth value. It operates on the `om:OMS` node to check if its `name` attribute has value equal to “R” and its `cd` attribute has value equal to “setname1”. The value of the second predicate represents the value of the expression; if its value is `true` then the third `OMS` node is selected.

Note that in complete MSQL queries the results of such path expressions are only used in the context of `WHERE` expressions to select result documents from a candidate set of documents; the result of an MSQL query always consists of full documents.

Operators

MSQL supports arithmetic, logical, and comparison operators. They are used inside predicates to perform arithmetic, logical, and comparison operations. Arithmetic operators include `+`, `-`, and `*`. Logical operators include `or`, `and`, and `not`. Comparison operators include `=`, `!=`, `<`, `<=`, `>`, and `>=`. In Use Case 1, the expression `(@name = ‘‘Z’’) and (@cd = ‘‘setname1’’)` in the last predicate uses the comparison operator = `and` the logical operator `and` to evaluate to a truth value. Operator `=` in the subexpression `(@name = ‘‘Z’’)` checks if the value of the attribute `name` equals to string “Z”. Also operator `=` in the second subexpression `(@cd = ‘‘setname1’’)` checks if the value of the attribute `cd` equals to string “setname1”. The operator `and` in the expression is used for the logical operation between the results of the two subexpressions.

Functions

In the context of a predicate, functions may be applied to the current node to extract information used in some operations. MSQL supports the functions stated in the rule.

```
<msqlFunction> ::= (EMPTY | COUNTS | CONTAINS | DOC | POSITION)
                   ‘‘(’’(<expression> | <expression>, <expression>)?‘‘)’’
```

- Function `empty()` returns *true* if its argument is an empty element.

- Function `count()` returns an integer value equivalent to the number of occurrences of its argument in the current node.

- Function `contains()` returns *true* if its first argument value contains as part of it its second argument.
- Function `doc()` returns the root node of the document whose name appear as its argument.
- Function `position()` returns the position (integer value) of the element, whose name precedes it in a path expression, in the child list of the current node.

Use Case 2. *Find all problems in "/GAMS/Arithmetic, error analysis/Integer" that have no precondition.*

```
SELECT EVERY problem
FROM /GAMS/Symbolic Computation
WHERE //body[empty(/pre-condition)]
```

In this Use Case, the `empty()` function takes the path expression `/pre-condition` as its argument and returns *true* if the node denoted by the path expression is empty. The query uses the `empty()` function to check if the "body" node has an empty "pre-condition" element node.

In Use Case 3 (shown below), the `contains()` function returns *true* if its first argument value (`@href` attribute value) contains as part of it its second argument (string "perseus"). The `doc()` function returns the root node of the document whose name is its argument value (`//implementation/@href`). The argument value is URI that is used as the name of the required document in the registry.

Conditional Expressions

Conditional expressions are used when the document to be returned depends on some condition. A conditional expression takes the following form:

```
<ifExpr> ::= IF <expression1> THEN <expression2> ELSE <expression3>
```

It is demonstrated in the following Use Case:

Use Case 3. *Find every service in "/GAMS/Linear Algebra" such that if it has an implementation it runs on a machine called perseus or its interface is on the said machine.*

```
SELECT EVERY service
FROM /GAMS/Linear Algebra WHERE
 if not (/service[empty(//implementation)])
 then
  let $d := doc(//implementation/@href) in
    $d/hardware[contains(@name, "perseus")]
 else //service-interface-description[
        contains(@href, "perseus")]
```

Use Case 3 shows a query that uses a conditional expression to decide if the current service document node has (`IsBasedOn`) an implementation. If this is the case, it takes the URI of such an implementation document and retrieves it from the registry and checks if this implementation is related to the machine `perseus`. If this is not the case, it checks (in the `else` clause) if the service has its interface on the said machine, i.e., `RunsOn` on the said machine.

Variable Bindings

An expression or a value that are used in more than one place in the same query can be bound to a variable so it does not need to be defined again. The `let` clause is used to bind a document to variable. It takes the following form:

```
<letExpr> ::= LET <var> := <expression1> IN <expression2>
```

The value of `expression1` is bound to variable `var` and it is substituted in `expression2`

In Use Case 3 the `let` clause binds a document to variable `d`. The value of variable `d` which is the bound document is then used in the path expression

```
$d/hardware[contains(@name, "perseus")].
```

The query in Use Case 3 aside from showing the expressiveness of MSQL in dealing with the MSDL content, it also reveals how MSQL utilizes the structure of the MSDL information model (see Figure 2) supported by the registry. It uses associations (e.g., `IsBasedOn`, `RunsOn`) among the entities of the model implicitly in the query.

Quantifiers

MSQL provides universal and existential quantifiers to test if every/some element in a document satisfies a certain condition. A quantified expression takes the following format:

```
<quantifiedExpr> ::= (SOME | EVERY) <var> IN <expression1>
                     SATISFIES <expression2>
```

It tests if `Some/Every` value of `expression1`, if substituted for `var` in `expression2` , makes the result of evaluation of `expression2` `true`. It is demonstrated in the following Use Case:

Use Case 4. *Find all problems in "/GAMS/Arithmetic, error analysis/Integer" in which the OpenMath content dictionary "sts" and the "mapsto" symbol are used in the same signature.*

```
SELECT EVERY problem FROM /GAMS/Arithmetic, error analysis/Integer
WHERE every $p in /problem satisfies
 some $s in $p//signature satisfies
  $s/om:OMOBJ/om:OMA/om:OMS[@cd = "sts"] and
  $s/om:OMOBJ/om:OMA/om:OMS[@name = "mapsto"]
```

The every quantifier requires all "problem" nodes to satisfy the "some" quantifier which checks if at least one signature satisfies the condition specified.

4 Formal Semantics of MSQL

In this section, we present a formal definition of MSQL using denotational semantics. Denotational semantics [29, 30] is a technique for rigorously specifying the semantics of a programming language. It maps language constructs to their meaning (called denotation) by specifying semantic functions that map elements of the language syntactic domains to elements of its semantic domains. Elements of the semantic domains are mathematical values such as numbers sets, functions, etc. This clarifies the intended meaning of the language constructs and provides a correct reference for implementation.

4.1 Format of the Denotational Definition

The denotational definition of MSQL consists of three parts: the abstract syntax definition of the language, the semantic algebras, and the valuation function. The valuation function connects the abstract syntax and the semantic algebras, more precisely it maps abstract syntax rules to values of semantic domains. It determines the meaning of a query executed in the context of a collection of MSDL documents published in the MathBroker registry by determining the set of documents retrieved from the registry.

4.2 Abstract Syntax

The abstract syntax consists of the syntactic domains together with a set of abstract rules which defines the MSQL grammar. The following are the syntactic domains. Letters and abbreviated string such as *Q*, *E*, *V*, *Qua*, etc. are typed variables defined over syntactic domains such as *Query*, *Expression*, *Variable*, *PathExpression*, etc.

$Q \in Query$
$Qua \in Quantifier$
$E \in Expression$
$V \in Variable$
$En \in Entity$
$Cl \in ClassificationNode$
$P \in PathExpression$
$PE \in RegularPathExpression$
$Sor \in Sort$
$F \in Function$
$EL \in ExpressionList$
$N \in Name$

The following is the abstract syntax of MSQL defined in the EBNF grammar format. The left hand side of a rule symbole *::=* represents a nonterminal (i.e., a variable as defined above) and the right hand side represents a terminal and/or a nonterminal with alternative(s). Key words are written in capital italic (e.g., *SELECT*, *FROM*, *PROBLEM*, etc) while variables are capital single letters or abbreviated strings starting with capitals (e.g., *Qua*, *Cl*, *E*, *V*, etc.)

// The MSQL query
Q ::= *SELECT Qua En FROM Cl WHERE E ORDERBY E Sor*

//Result selection
Qua ::= *EVERY*
| *SOME*

// Entity types of the information model
En ::= *PROBLEM*
| *ALGORITHM*
| *IMPLEMENTATION*
| *MACHINE*
| *SERVICE*

// Sorting is ascending or descending order
Sor ::= *ASCENDING*
| *DESCENDING*

// An MSQL expression which evaluates to a value

E ::=
V // Variable
| P // Path expression
| $F(EL)$ // Function and its parameters as an expression list.
| *SOME V IN E SATISFIES E* // Existential quantification.
| *EVERY V IN E SATISFIES E* // Universal quantification.
| *IF E THEN E ELSE E* // Conditional.
| *LET V = E IN E* // Variable binding.
| $E + E$ // Represents arithmetic expressions, e.g., $-$ and $*$
| $E = E$ // Comparative operators $\leq$, $<$, $\geq$, $>$, and $\neq$
| *E AND E* // Represents logical connectives, e.g., OR and NOT.

// Functions
F ::= *COUNT* // counts the number of elements in a given node.
| *CONTAINS* // checks if a string contains another string.
| *EMPTY* // checks if a given element is empty.
| *POSITION* // returns the position of an element in a list.
| *DOC* // returns the document with the given argument name.

// Expression List as function parameters.
EL ::= E
| EL, E

// Path expression
P ::= PE
| P PE

// Regular path expression
PE ::= .
| /N // path step.
| //N // deep path step.
| @N // attribute step.
| [E] // predicate.

4.3 Semantic Algebras

The fundamental concept in semantic algebras is a semantic domain. A domain plus its operations constitutes a semantic algebra.

Semantic Domains

The semantic domains for MSQL are constructed from simple sets such as integer values, boolean values, string values, and sets. More complicated domains are constructed from simpler ones using, e.g., disjoint union. These provide an adequate formalism for the purpose of specifying MSQL semantics without requiring complex mathematical constructions.

The main semantic domains and their semantic algebras are as follows:

I. **Truth Values**
Domain $b \in Bool$
Operations
$true, false : Bool$

$or, and : Bool \times Bool \rightarrow Bool$
$not : Bool \rightarrow Bool$

II. **Integer Numbers**

$Domain\ i \in Int$
Operations
$0\ ..\ 9 : Int$
$+,\ -,\ * : Int \times Int \rightarrow Int$
$=,\ \neq,\ >,\ \geqslant,\ <,\ \leqslant : Int \times Int \rightarrow Bool$

III. **String**

$Domain\ s \in Str$
Operations
$a\ ..\ z,\ A\ ..\ Z,\ 0\ ..\ 9,\ .,\ \_ : Str$
$empty : Str$
$length : Str \rightarrow Int$
$concat : Str \times Str \rightarrow Str$
$isSubstr =: Str \times Str \rightarrow Bool$ // Checks if the second Str is a substring of the first.
$=,\ <,\ > : Str \times Str \rightarrow Bool$ // String alphabetical comparison operations.

IV. **List**

The *List* domain models a list of nodes that are produced or used by the query. It offers a set of operations for constructing and manipulating a list of nodes.

$Domain\ l \in List$
Operations
$newList : List$
$isEmpty : List \rightarrow Bool$
$length : List \rightarrow Int$
$append : List \times Node \rightarrow List$ // appends a node to the list.
$get : List \times Int \rightarrow Node$ // returns the element node at position i.
$\in : List \times Node \rightarrow Bool$ // checks if the specified node is an element in the list.
$concat : List \times List \rightarrow List$ // concatenates two lists.
$remove : List \times Node \rightarrow List$ // removes the specified node from the list.
$sort : List \times CF \rightarrow List$ // CF is an ordering function that orders the values representing nodes in the list.

V. **CF**

The *CF* domain models an ordering function that compares two nodes. It is used for sorting the nodes in the query result.

$Domain\ cf \in CF = Node \times Node \rightarrow Bool$

VI. **Value**

The *Value* domain is a disjoint union of the Boolean, Integer, String, List, and Message domains. It models the possible values encountered during the evaluation of a query. Its "*in...*" operations map a given value to its respective sub-domain. Its "*is...*" operations check if a given value belongs to a certain domain.

$Domain\ v \in Value = Bool + Int + Str + List + Message$
Operations
$inBool : Bool \rightarrow Value$
$inInt : Int \rightarrow Value$

$inStr : Str \rightarrow Value$
$inList : List \rightarrow Value$
$inMessage : Message \rightarrow Value$
$isBool : Value \rightarrow Bool$ // checks if the given value is a boolean value.
$isInt : Value \rightarrow Bool$
$isStr : Value \rightarrow Bool$
$isList : Value \rightarrow Bool$
$isMessage : Value \rightarrow Message$
$boolCheck : Value \rightarrow Bool$
$boolCheck(v) =$ **cases** v **of**
$isBool(b) \rightarrow true$
$isList(l) \rightarrow true$
$otherwise\ false$

$toBool : Value \rightarrow Bool$
$toBool(v) =$ **cases** v **of**
$isBool(b) \rightarrow b$
$isList(l) \rightarrow not\ isEmpty(l)$
$otherwise\ false$

$<: Value \times Value \rightarrow Value$
$>: Value \times Value \rightarrow Value$
$v_1 > v_2 =$ **cases** v_1 **of**
$isInt(i_1) \rightarrow$ (**cases** v_2 **of**
$isInt(i_2) \rightarrow\ inBool(i_1\ >\ i_2)$
$otherwise\ inMessage(ERROR)$)
$isStr(s_1) \rightarrow$ (**cases** v_2 **of**
$isStr(s_2) \rightarrow\ inBool(s_1\ >\ s_2)$
$otherwise\ inMessage(ERROR)$)
$otherwise\ inMessage(ERROR)$

VII. **Sequence**

The *Sequence* domain is defined like the *List* domain but it has *Value* elements and it uses some additional operations.

$Domain\ seq \in Seq$
$Operations$
$newSeq : Seq$
$isEmpty : Seq \rightarrow Bool$
$length : Seq \rightarrow Int$
$append : Seq \times Value \rightarrow Seq$
$get : Seq \times Int \rightarrow Value$ // returns the value element at position i.
$\in : Seq \times Value \rightarrow Bool$

VIII. **Registry**

The *Registry* domain models a registry as a mapping from classification nodes and entity types to sets of documents. The first operation upon the registry is to query it based on the classification node and entity type and retrieve a set of documents. The second operation is used to query the registry based on the entity name only retrieving that document with the given name.

$Domain\ r \in Registry$
$Operations$
$rQuery : Classification Node \times Entity \times Registry \rightarrow \mathbb{P}(Doc)$
$rQuery : Name \times Registry \rightarrow Doc$

IX. **Doc**
The *Doc* domain models an MSDL document retrieved from the registry. The *docNode* operation converts a document to a node that is to be used by the query operations.

$Domain\ doc \in Doc$
$Operations$
$docNode : Doc \rightarrow Node$

X. **Node**
The *Node* domain models a document node. It uses a set of operations to access and decide on the elements contained in a node. The *name* operation returns the string name of the node. The *position* operation returns an integer value equivalent to the node's positions relative to its parent. The *children* operation returns as a list the children of the given node. The *attributes* operation returns as a list the attributes of the given node. The operations *attrName* and respectively *attrValue* return the name and the value of the given attribute.

$Domain\ n \in Node$
$Operations$
$name : Node \rightarrow String$
$position : Node \rightarrow Int$
$children : Node \rightarrow List$
$attributes : Node \rightarrow List$
$attrName : Node \rightarrow String$
$attrValue : Node \rightarrow Value$

XI. **Declaration**
The *Declaration* domain models a declaration where declared variables and their values are stored. The operation *put* declares a variable by giving it a value and *lookup* returns the value of the given variable from the declaration.

$Domain\ d \in Decl = V \rightarrow Value$
$Operations$
$newDecl : Decl$
$put : Decl \times Variable \times Value \rightarrow Decl$
$lookup : Decl \times Variable \rightarrow Value$

XII. **Message**
The *Message* domain models a message that is returned if something goes wrong during the query evaluation. It consists of one element, the ERROR element.

$Domain\ m \in Message = \{ERROR\}$

XIII. **Name**
The *Name* domain models a name of, for example, an entity or a path step.

$Domain\ na \in Name$
$Operations$

$isName : String \rightarrow Bool$ // checks if a string is a valid name.
$toName : String \rightarrow Name$ // returns a name out of a string.

4.4 Semantic Functions

Semantic functions, also called valuation functions, specify the actual meaning of each syntactic construct. They map every member of the syntactic domain into a member of the semantic domain. There is a function for each rule in the abstract syntax. All the syntactic arguments are enclosed in $[\![\ ]\!]$ to show they are members of a syntactic domain. Most of the semantic functions take four arguments: a syntactic string, a declaration *d*, an MSDL document node *n*, and the registry *r*. An example semantic equation is

$$\mathbf{P}[\![PE]\!]\ d\ n\ r = \mathbf{PE}[\![PE]\!]\ d\ n\ r$$

In this equation, the meaning of the syntactic expression PE is defined by the semantic function **P**. **P** is a function that takes four arguments: the syntactic expression PE, a declaration *d*, a node *n*, and the registry *r*. The result is defined in terms of another sematic function **PE**.

In the next sections we define the following semantic functions:

- The query evaluation function

  $\mathbf{Q} : Query \rightarrow Registry \rightarrow \mathbb{P}(Doc) + Message$

- The expression evaluation function

  $\mathbf{E} : Expression \rightarrow Decl \rightarrow Node \rightarrow Registry \rightarrow Value$

- The MSQL-Function evaluation function

  $\mathbf{F} : Function \rightarrow Node \rightarrow Registry \rightarrow Seq \rightarrow Value$

- The expression list evaluation function

  $\mathbf{EL} : ExpressionList \rightarrow Decl \rightarrow Node \rightarrow Registry \rightarrow Seq$

- The name (of step) evaluation function

  $\mathbf{N} : Name \rightarrow Str$

- The path expression evaluation function

  $\mathbf{P} : PathExpression \rightarrow Decl \rightarrow Node \rightarrow Registry \rightarrow \mathbb{P}(Node)$

- The regular path expression evaluation function

  $\mathbf{PE} : RegularPathExpression \rightarrow Decl \rightarrow Node \rightarrow Registry \rightarrow \mathbb{P}(Node)$

4.4.1 Query Function

We start with the function **Q** which takes a query as its first argument and the registry as its second argument. It uses the operation *rQuery* to query the registry for candidate documents satisfying the specified *classificationNode* and *Entity* type. It initializes a new declaration for use in subsequent functions. It checks first if the list of candidate documents is not empty and has at least some document satisfying the query criteria specified in the expression E_1, then based on the *SOME/EVERY* clause it returns some document or every document satisfying the criteria specified in E_1. If the *EVERY* clause is specified, it also sorts the resulting documents

using an ordering function *cf*. The function *cf* operates on the list of the resulting documents to sort it in *ASCENDING/DESCENDING* order based on the criteria specified in expression E of the query.

$$\mathbf{Q} : Query \rightarrow Registry \rightarrow \mathbb{P}(Doc) + Message$$

$$\begin{array}{l}
\mathbf{Q}[\![SELECT\ Qua\ En\ FROM\ Cl\ WHERE\ E_1\ ORDERBY\ E_2\ Sor]\!]r = \\
\quad \textbf{let}\ D = rQuery(Cl, En, r) \\
\qquad d = newDecl() \\
\quad \textbf{in if}\ \forall doc \in D : isBool(\mathbf{E}[\![E_1]\!]\ d\ docNode(doc)\ r) \\
\qquad \textbf{then if}\ Qua = [\![SOME]\!] \\
\qquad\qquad \textbf{then if}\ \exists doc \in D : toBool(\mathbf{E}[\![E_1]\!]\ d\ docNode(doc)\ r) \\
\qquad\qquad\qquad \textbf{then}\ \{such\ doc : doc \in D\ and\ toBool(\mathbf{E}[\![E_1]\!]\ d\ docNode(doc)\ r)\} \\
\qquad\qquad\qquad \textbf{else}\ \{\} \\
\qquad\qquad \textbf{else if}\ Qua = [\![EVERY]\!] \\
\qquad\qquad\qquad \textbf{then}\ let\ l = \{doc \in D\ \mid toBool(\mathbf{E}[\![E_1]\!]\ d\ docNode(doc))\ r\} \\
\qquad\qquad\qquad\quad \textbf{in if}\ Sor = [\![DESCENDING]\!] \\
\qquad\qquad\qquad\qquad \textbf{then}\ sort(l,\ \lambda(doc_1,\ doc_2). \\
\qquad\qquad\qquad\qquad\qquad (\mathbf{E}[\![E_2]\!]\ d\ docNode(doc_1)\ r > \mathbf{E}[\![E_2]\!]\ d\ docNode(doc_2)\ r)) \\
\qquad\qquad\qquad\qquad \textbf{else}\ sort(l,\ \lambda(doc_1,\ doc_2). \\
\qquad\qquad\qquad\qquad\qquad (\mathbf{E}[\![E_2]\!]\ d\ docNode(doc_1)\ r < \mathbf{E}[\![E_2]\!]\ d\ docNode(doc_2)\ r)) \\
\qquad\qquad\qquad \textbf{else}\ \{\} \\
\qquad \textbf{else}\ inMessage(ERROR)
\end{array}$$

4.4.2 Expression Function

The function **E** maps an expression *Expression* to a value in the domain *Value*. Arguments to **E** are the declaration *Decl* and the current document node *Node*.

$$\mathbf{E} : Expression \rightarrow Decl \rightarrow Node \rightarrow Registry \rightarrow Value$$

- The equation for $\mathbf{E}[\![V]\!]$ looks up and returns the value of variable V from the declaration.

  $$\mathbf{E}[\![V]\!]\ d\ n\ r = lookup(d,\ [\![V]\!])$$

- The equation of $\mathbf{E}[\![P]\!]$ states that its value follows from the evaluation of function **P** which is defined latter for path expressions.

  $$\mathbf{E}[\![P]\!]\ d\ n\ r = \mathbf{P}[\![P]\!]\ d\ n\ r$$

- The equation of $\mathbf{E}\ [\![F(EL)]\!]$ performs MSQL function evaluation. It evaluates the argument list expressions to a sequence of values and then calls the semantic function **F** to evaluate function F with the sequence of values as argument.

  $$\begin{array}{l}
  \mathbf{E}\ [\![F(EL)]\!]\ d\ n\ r = \\
  \quad \textbf{let}\ seq = \mathbf{EL}[\![EL]\!]\ d\ n\ r \\
  \quad\quad \textbf{in}\ \mathbf{F}[\![F]\!]\ n\ r\ seq
  \end{array}$$

- The equation for the existential quantification $\mathbf{E}[\![SOME\ x\ in\ E_1\ SATISFIES\ E_2]\!]$ tests if there is a value in the sequence we get by evaluating E_1, if substituted for x in E_2, makes the result of the evaluation of E_2 *true*.

$$
\begin{array}{l}
\mathbf{E}[\![SOME\ x\ in\ E_1\ SATISFIES\ E_2]\!]\ d\ n\ r = \\
\quad \textbf{let}\ e_1 = \mathbf{E}[\![E_1]\!]\ d\ n\ r \\
\quad\quad \textbf{in if}\ not\ isList(e_1) \\
\quad\quad\quad\quad \textbf{then}\ inMessage(ERROR) \\
\quad\quad\quad\quad \textbf{else if}\ (\exists v \in e_1 : boolCheck(\mathbf{E}[\![E_2]\!]\ [[\![x]\!] \mapsto v]d\ n\ r)\ and \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad toBool(\mathbf{E}[\![E_2]\!]\ [[\![x]\!] \mapsto v]d\ n\ r)) \\
\quad\quad\quad\quad\quad\quad \textbf{then}\ inBool(true) \\
\quad\quad\quad\quad\quad\quad \textbf{else}\ inBool(false)
\end{array}
$$

- The equation for the universal quantification $\mathbf{E}[\![EVERY\ x\ in\ E_1\ SATISFIES\ E_2]\!]$ tests if every value in the sequence we get by evaluating E_1, if substituted for x in E_2, makes the result of the evaluation of E_2 *true*.

$$
\begin{array}{l}
\mathbf{E}[\![EVERY\ x\ in\ E_1\ SATISFIES\ E_2]\!]\ d\ n\ r = \\
\quad \textbf{let}\ e_1 = \mathbf{E}[\![E_1]\!]\ d\ n\ r \\
\quad\quad \textbf{in if}\ not\ isList(e_1) \\
\quad\quad\quad\quad \textbf{then}\ inMessage(ERROR) \\
\quad\quad\quad\quad \textbf{else if}\ (\forall v \in e_1 : boolCheck(\mathbf{E}[\![E_2]\!]\ [[\![x]\!] \mapsto v]d\ n\ r)\ and \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad toBool(\mathbf{E}[\![E_2]\!]\ [[\![x]\!] \mapsto v]d\ n\ r)) \\
\quad\quad\quad\quad\quad\quad \textbf{then}\ inBool(true) \\
\quad\quad\quad\quad\quad\quad \textbf{else}\ inBool(false)
\end{array}
$$

- The equation of $\mathbf{E}[\![if\ E_1\ then\ E_2\ else\ E_3]\!]$ evaluates E_1 and if it is *true* it considers E_2 for evaluation otherwise it considers E_3 for evaluation.

$$
\begin{array}{l}
\mathbf{E}[\![if\ E_1\ then\ E_2\ else\ E_3]\!]\ d\ n\ r = \\
\quad \textbf{let}\ e = \mathbf{E}[\![E_1]\!]\ d\ n\ r \\
\quad\quad \textbf{in if}\ boolCheck(e) \\
\quad\quad\quad\quad \textbf{then let}\ v = toBool(e) \\
\quad\quad\quad\quad\quad\quad \textbf{in cases}\ v\ \textbf{of} \\
\quad\quad\quad\quad\quad\quad\quad\quad true \rightarrow \mathbf{E}[\![E_2]\!]\ d\ n\ r \\
\quad\quad\quad\quad\quad\quad\quad\quad false \rightarrow \mathbf{E}[\![E_3]\!]\ d\ n\ r \\
\quad\quad\quad\quad \textbf{else}\ inMessage(ERROR)
\end{array}
$$

- The equation of $\mathbf{E}[\![let\ x = E_1\ in\ E_2]\!]$ evaluates E_1 and binds the resulting value to variable x in the declaration d. This value is substituted for every occurrence of variable x in the evaluation of E_2.

$$
\begin{array}{l}
\mathbf{E}[\![let\ x = E_1\ in\ E_2]\!]\ d\ n\ r = \\
\quad \textbf{let}\ e_1 = \mathbf{E}[\![E_1]\!]\ d\ n\ r \\
\quad\quad \textbf{in}\ \mathbf{E}[\![E_2]\!]\ [[\![x]\!] \mapsto e_1]d\ n\ r
\end{array}
$$

- The equation of $\mathbf{E}[\![E_1 + E_2]\!]$ performs the addition on the two integer numbers resulting from the evaluation of $\mathbf{E}[\![E_1]\!]$ and $\mathbf{E}[\![E_2]\!]$.

$$
\begin{array}{l}
\mathbf{E}[\![E_1 + E_2]\!]\ d\ n\ r = \\
\quad \textbf{let}\ e_1 = \mathbf{E}[\![E_1]\!]\ d\ n\ r \\
\qquad\ e_2 = \mathbf{E}[\![E_2]\!]\ d\ n\ r \\
\quad \textbf{in cases}\ e_1\ \textbf{of} \\
\qquad\ isInt(i_1) \rightarrow \textbf{cases}\ e_2\ \textbf{of} \\
\qquad\qquad\qquad\qquad isInt(i_2) \rightarrow inInt(i_1 + i_2) \\
\qquad\qquad\qquad\qquad otherwise\ inMessage(ERROR) \\
\qquad\ otherwise\ inMessage(ERROR)
\end{array}
$$

- The equation of $\mathbf{E}[\![E_1\ =\ E_2]\!]$ evaluates the comparison operator =. It returns a boolean value, if both argument expressions evaluate to the same boolean, integer, or string and an error otherwise.

$$
\begin{array}{l}
\mathbf{E}[\![E_1\ =\ E_2]\!]\ d\ n\ r = \\
\quad \textbf{let}\ e_1 = \mathbf{E}[\![E_1]\!]\ d\ n\ r \\
\qquad\ e_2 = \mathbf{E}[\![E_2]\!]\ d\ n\ r \\
\quad\ \textbf{in cases}\ e_1\ \textbf{of} \\
\qquad\ isBool(b_1) \rightarrow (\textbf{cases}\ e_2\ \textbf{of} \\
\qquad\qquad\qquad\qquad isBool(b_2) \rightarrow inBool(b_1 = b_2) \\
\qquad\qquad\qquad\qquad otherwise\ inMessage(ERROR)) \\
\qquad\ isInt(i_1) \rightarrow (\textbf{cases}\ e_2\ \textbf{of} \\
\qquad\qquad\qquad\qquad isInt(i_2) \rightarrow inBool(i_1 = i_2) \\
\qquad\qquad\qquad\qquad otherwise\ inMessage(ERROR)) \\
\qquad\ isStr(s_1) \rightarrow (\textbf{cases}\ e_2\ \textbf{of} \\
\qquad\qquad\qquad\qquad isStr(s_2) \rightarrow inBool(s_1 = s_2) \\
\qquad\qquad\qquad\qquad otherwise\ inMessage(ERROR)) \\
\qquad\ otherwise\ inMessage(ERROR)
\end{array}
$$

- The equation of $\mathbf{E}[\![E_1\ AND\ E_2]\!]$ evaluates the logical connective *AND*. It returns a boolean value or an error if both of its argument expressions are not evaluated to either a boolean or a list that is not empty.

$$
\begin{array}{l}
\mathbf{E}[\![E_1\ AND\ E_2]\!]\ d\ n\ r = \\
\quad \textbf{let}\ e_1 = \mathbf{E}[\![E_1]\!]\ d\ n\ r \\
\qquad\ e_2 = \mathbf{E}[\![E_2]\!]\ d\ n\ r \\
\quad\ \textbf{in if}\ boolCheck(e_1) \\
\qquad\ \textbf{then if}\ boolCheck(e_2) \\
\qquad\qquad\qquad\ \textbf{then}\ toBool(e_1)\ and\ toBool(e_2) \\
\qquad\qquad\qquad\ \textbf{else}\ inMessage(ERROR) \\
\qquad\ \textbf{else}\ inMessage(ERROR)
\end{array}
$$

4.4.3 MSQL-Function Function

Function **F** maps an MSQL function *Function* that has a sequence of values as an argument to a value in the domain *Value*. It is called by function $\mathbf{E}\ [\![F(EL)]\!]$ which passes to it the sequence of values as argument.

$\mathbf{F} : Function \rightarrow Node \rightarrow Registry \rightarrow Seq \rightarrow Value$

- The equation of the $\mathbf{F}[\![POSITION]\!]$ function returns an integer value corresponding to the position of the current node in node list. It uses the operation *position* defined for the *Node* domain to determine the position of the node in the list.

$$\begin{array}{l}\mathbf{F}[\![POSITION]\!]\ n\ r\ seq = \\ \quad \textbf{if}\ isEmpty(seq) \\ \quad \textbf{then}\ inInt(position(n)) \\ \quad \textbf{else}\ inMessage(ERROR)\end{array}$$

- $\mathbf{F}[\![CONTAINS]\!]$ checks if the first string evaluated argument E_1 contains the second string evaluated argument E_2. It uses the operation *isEmpty* which is defined in the **List** domain.

$$\begin{array}{l}\mathbf{F}[\![CONTAINS]\!]\ n\ r\ seq = \\ \quad \textbf{if}\ length(seq) \neq 2 \\ \quad \textbf{then}\ inMessage(ERROR) \\ \quad \textbf{else cases}\ get(seq,\ 1)\ \textbf{of} \\ \qquad isStr(s_1) \rightarrow (\textbf{cases}\ get(seq,\ 2)\ \textbf{of} \\ \qquad\qquad isStr(s_2) \rightarrow\ inBool(isSubstr(s_1,\ s_2)) \\ \qquad\qquad otherwise\ inMessage(ERROR)) \\ \qquad otherwise\ inMessage(ERROR)\end{array}$$

- $\mathbf{F}[\![COUNT]\!]$ returns as integer the length of the current node list. It uses the operation *length* which is defined in the *List* domain.

$$\begin{array}{l}\mathbf{F}[\![COUNT]\!]\ n\ r\ seq = \\ \quad \textbf{if}\ length(seq) \neq 1 \\ \quad \textbf{then}\ inMessage(ERROR) \\ \quad \textbf{else cases}\ get(seq,\ 1)\ \textbf{of} \\ \qquad isList(l) \rightarrow inInt(length(l)) \\ \qquad otherwise\ inMessage(ERROR)\end{array}$$

- $\mathbf{F}[\![EMPTY]\!]$ checks if the current node list is empty. It uses the operation *isEmpty* which is defined in the **List** domain.

$$\begin{array}{l}\mathbf{F}[\![EMPTY]\!]\ n\ r\ seq = \\ \quad \textbf{if}\ length(seq) \neq 1 \\ \quad \textbf{then}\ inMessage(ERROR) \\ \quad \textbf{else cases}\ get(seq,\ 1)\ \textbf{of} \\ \qquad isList(l) \rightarrow isEmpty(l) \\ \qquad otherwise\ inMessage(ERROR)\end{array}$$

- $\mathbf{F}[\![DOC]\!]$ retrieves a document from the registry with the given string argument as the name of the document. It uses the *rQuery* operation defined in the *Registry* domain to query the registry for the document with the given name.

$$\begin{array}{l}\mathbf{F}[\![DOC]\!]\ n\ r\ seq = \\ \quad \textbf{if}\ length(seq) \neq 1\end{array}$$

then $inMessage(ERROR)$
else cases $get(seq,\ 1)$ **of**
$isStr(s) \rightarrow$ **cases** s **of**
$isName(s) \rightarrow\ rQuery(toName(s),\ r)$
$otherwise\ inMessage(ERROR)$
$otherwise\ inMessage(ERROR)$

4.4.4 Expression List Function

Function **EL** maps an expression list to a sequence of values.

$\mathbf{EL} : ExpressionList \rightarrow Decl \rightarrow Node \rightarrow Registry \rightarrow Seq$

- The equation of $\mathbf{EL}[\![EL,\ E]\!]$ evaluates the expression list by evaluating each expression using the function $\mathbf{EL}[\![E]\!]$ and appending it to the sequence of values.

$\mathbf{EL}[\![EL,\ E]\!]\ d\ n\ r =$
let $seq = \mathbf{E}[\![EL]\!]\ d\ n\ r$
$e = \mathbf{E}[\![E]\!]\ d\ n\ r$
in $append(seq, e)$

$\mathbf{EL}[\![E]\!]\ d\ n\ r =$
let $e = \mathbf{E}[\![E]\!]\ d\ n\ r$
in $append(newSeq, e)$

4.4.5 Name Function

Function **N** maps a *Name* to a value in the string domain.

$\mathbf{N} : Name \rightarrow Str$

4.4.6 Regular Path Expression Function

Function **PE** maps a regular path expression PE to the power set nodes.

$\mathbf{PE} : RegularPathExpression \rightarrow Decl \rightarrow Node \rightarrow Registry \rightarrow \mathbb{P}(Node)$

- $\mathbf{PE}[\![.]\!]$ returns a list containing the the context node n itself.

$\mathbf{PE}[\![.]\!]\ d\ n\ r = \{n\}$

- $\mathbf{PE}[\![/N]\!]$ returns a list of nodes that are children of the current node n with the tag name (string value of N).

$\mathbf{PE}[\![/N]\!]\ d\ n\ r = \{m \mid m \in\ children(n)\ and\ name(m) = \mathbf{N}[\![N]\!]\}$

- $\mathbf{PE}\ [\![//N]\!]$ returns a list of nodes that are children of the current node n with tag names (string value of N) no matter where they are in the node hierarchy.

$\mathbf{PE}[\![//N]\!]\ d\ n\ r = p\ \mathbf{N}[\![N]\!]\ d\ n\ r$

$p : String \rightarrow Decl \rightarrow Node \rightarrow Registry \rightarrow \mathbb{P}(Node)$

$p(s,\ d,\ n\ r) =$
$\{m_2 : (m_2 \in children(n)\ and\ name(m_2) = s)\ or$
$(\exists m_1 : m_1 \in children(n)\ and\ name(m_1) \neq s\ and\ m_2 \in p(s,\ d,\ m_1))\}$

The meaning of this recursive function definition is defined as usual by fixed point construction [29].

- **PE**$[\![@N]\!]$ returns the attribute value of the given attribute with the given name (the string value of N).

  PE$[\![@N]\!]\ d\ n\ r =$
  $\{v \mid \exists a :\ a \in attributes(n)\ and\ attrName(a) = \mathbf{N}[\![N]\!]\ and\ attrValue(a) = v\}$

- **PE**$[\![\ [E]\ ]\!]$ returns a boolean value resulting from the evaluation of the predicate expression $[E]$.

  PE$[\![\ [E]\ ]\!]\ d\ n\ r =$
  let $e =$ **E**$[\![E]\!]\ d\ n\ r$
  in if $boolCheck(e)\ and\ toBool(e)$
  then $\{n\}$
  else $\{\}$

4.4.7 Path Function

Function **P** maps a path expression *PathExpression* to a power set of nodes.

P : $PathExpression \rightarrow Decl \rightarrow Node \rightarrow Registry \rightarrow \mathbb{P}(Node)$

- The equation of **P**$[\![PE]\!]$ evaluates the **PE** function with argument PE as indicated before.

  P$[\![PE]\!]\ d\ n\ r =$ **PE**$[\![PE]\!]\ d\ n\ r$

- The equation of **P**$[\![P\ PE]\!]$ applies the **P** function on the expression P and if the resulting value is a list of a single node then the function **PE** is applied on this node and the result is returned. If the result is a list of more than one node, then **PE** is evaluated on each of them and the results are gathered in the power set q.

  P$[\![P\ PE]\!]\ d\ n\ r =$
  let $p =$ **P**$[\![P]\!]\ d\ n\ r$
  in $\bigcup\{$**PE**$[\![PE]\!]\ d\ n\ r : n \in p\}$

5 Implementation of MSQL

In this section, we present a prototype implementation of MSQL which is based on the denotational semantics of the language. The advantage of using this approach is that it provides a reference for implementation. We describe the MSQL architecture which represents a high-level overview of the underlying implementation. We discuss some of the most crucial implementation decisions such as the parser implementation and the employed data model for the representation of an MSDL document when processed by a query. We also present the resulting API and how it can be used by a client program.

5.1 The MSQL Architecture

Figure 3 provides a high-level overview of the MSQL architecture which consists of the following parts:

- The *MSQL Engine* which constitutes the querying functionality.
- The *MathBroker Registry* where MSDL documents are published.
- The *Reasoner* which handles the part of a query that needs reasoning (currently being implemented).

A client application sends a query to the MSQL engine and receives a set of MSDL descriptions. These descriptions are either returned to the user or processed further for specific tasks, e.g., to access the service having the resulting description.

The MSQL Engine consists of the following components which are designed according to the query structure explained in Section 3.4:

- **The Query Processor** which receives the query, divides it into processable parts, and hands each part to its corresponding component.
- **The Registry Handler** which receives from the processor the query part needed to retrieve MSDL documents from the registry based on their types and classifications.
- **The Expression Evaluator** which evaluates the expression part of a query against the documents retrieved from the registry, filters them, and forwards those passing the filter to the Result Quantifier and Sorter component.
- **The Reasoner Interface** which receives from the processor the query part that needs reasoning and sends it to the reasoner.
- **The Result Quantifier and Sorter** which decides whether to return some or every queried document as a result and whether to sort the resulting documents.

In this architecture, an MSQL query received from a client is processed as follows:

- The query is parsed according to MSQL grammar and transformed into an abstract syntax tree.
- The engine connects to the registry and invokes the "MathBroker Query Manager" which is part of the registry API. It queries the registry for candidate documents based on the type of document required and based on the classification node under which the required document is classified. It returns to the engine the collection of candidate MSDL documents.
- The Expression Evaluator then evaluates the condition expression of the query against each candidate document and, if the document satisfies the condition, adds it to the collection of resulting documents.
- The Result Quantifier and Sorter then quantifies and orders the resulting documents depending on the kind of query (see the next section).

The engine then returns the resulting MSDL documents to the client application.

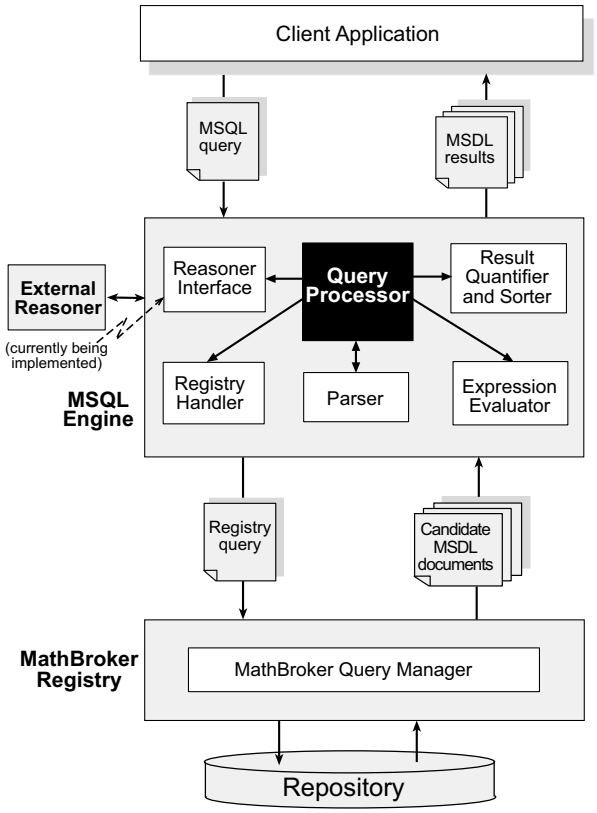

Figure 3: The MSQL Architecture

5.2 Semantic-Based Implementation

In Section 4 we presented a formal specification of MSQL using Denotational Semantics. This semantics provides a reference for the correctness of implementation of the language. Thus we have attempted, with some minor exceptions, to follow in the implemented methods the definitions of the semantics as closely as possible. One such minor exception is in the implementation of the *Query* function **Q** where we did not check that every retrieved document satisfy the stated boolean condition before evaluating the query on it.

The implementation consists of a set of evaluation classes with a set of methods each corresponding to each semantic function. The signature of a method corresponds to the signature of the semantic function. For example, the semantic function

$$\mathbf{E}[\![V]\!]\ d\ n\ r = lookup(d,\ [\![V]\!])$$

is implemented by the Java method

```
static private Value evaluateVariableExpr(ChildAST expr,
                                          Declaration declaration,
                                          Node node) throws MsqlException {
  VarExpr varExpr = (VarExpr)expr;
  return declaration.lookup(varExpr);
}
```

Each semantic domain (see Section 4) is implemented as a separate class providing as methods the semantic functions of this domain.

5.3 Implementation Details

We state the most important design decisions which influenced the implementation of MSQL.

5.3.1 Lexer, Parser, and Tree Walker Implementation

We used ANTLR (ANother Tool for Language Recognition) [27] to implement the grammar of the lexer and the parser of MSQL. ANTLR is a lexer/parser generator tool that lets one define language grammars in a EBNF (Extended Backus-Naur) form from which it generates lexers, parsers, and data representation of Abstract Syntax Trees (ASTs). An AST is constructed during a parsing phase and used as the internal representation of an input statement.

We expressed the MSQL grammar in ANTLR syntax making the necessary changes to comply with the rules of grammar definition in ANTLR.

MSQL Lexer The role of the lexer, known also as tokenizer, is to receive a query string and construct the corresponding language tokens. The lexer grammar expressed in ANTLR syntax can be found in Appendix A.

MSQL Parser The role of the parser is to check if the generated tokens form a valid sentence according to the language syntax and then construct the AST. The parser grammar expressed in ANTLR syntax can be found in Appendix A.

MSQL Tree Walker The resulting AST of MSQL grammar belongs to what is called in ANTLR terminology a "heterogeneous AST", i.e., AST in which the nodes can have various types (corresponding to the various syntactic domains of the abstract syntax). In order to traverse (walk) an AST, we implemented a class for each kind of node in it.

The lexer, parser, and tree walker API can be found in Appendix D as part of the MSQL API.

5.3.2 Modeling MSDL documents as DOM objects

Like in XML, an MSDL document is modeled as a tree of nodes (see Section 3.3.) A node can be a root, an element node (element), an attribute, or a text. We used the DOM (Document Object Model) [8] to represent such a tree by an object of class *Node* with an interface that allows us to access the required parts of the tree.

5.3.3 MSQL API packages

We structured the implementation in packages:

1. The *msqlParser* package contains the lexer and parser of MSQL.
2. The *TreeWalker* package contains all the classes needed for walking the AST generated by the parser.
3. The *msqlSemantics* package (MSQL Engine) contains the implementation of the denotational semantics of MSQL.

The complete API of MSQL can be found in Appendix D

5.4 Using MSQL API

The interface `msqlQuery` represents the starting point for using the MSQL API. Its use is outlined as follows:

```
MsqlQuery msqlQuery = new MsqlQueryImpl();
MathBrokerConnection connection = msqlQuery.makeConnection(connectionProps);
ChildAST queryTree = msqlQuery.parseQuery(queryString);
Collection resultsCollection = msqlQuery.performQuery(queryTree, connection);
```

A connection is made to the registry through the `makeConnection` method. A received MSQL query is parsed using the `parseQuery` method. The query is processed and the results are returned a collection using the `performQuery` method. Returning a collection means that all resulting documents are returned to the user.

An alternative for using the API is:

```
MsqlQuery msqlQuery = new MsqlQueryImpl();
MathBrokerConnection connection = msqlQuery.makeConnection(connectionProps);
ChildAST queryTree = msqlQuery.parseQuery(queryString);
Iterator resultsCollection = msqlQuery.iterateQuery(queryTree, connection);
```

The method `iterateQuery` is used instead of `performQuery`. It allows the client to iteratively ask for one document satisfying a query after the other. If the client is satisfied with some result, then the query needs not be performed in the rest of candidate documents.

A complete client application demonstrating the use of the API can be found in Appendix B.

6 Related Work

The ebXML registry [18] [17] [16] respectively the Mathbroker registry [3] [4] do not address content-based querying for published objects. A registry object is known by its metadata such as its name, unique identifier, description, classifications, associations, and/or repository item which holds the intended content. The content has no way of being searched or queried by the querying facility of the registry. These facilities only search the object's metadata and may return the repository item as a result of the search. Although this facility is implemented in the MathBroker registry, we realized from the first stages of our registry development that they are insufficient in our case and should be extended to querying the contents of published mathematical objects.

In [2], we investigated a set of tools and query languages for storing and querying XML-content to gain a sufficient background for deciding on how to query MSDL content published in the MathBroker registry. These languages range from such that provide simple node finding capabilities based on path expressions to more comprehensive ones that support processing, transformation, and querying tasks of XML documents. QUILT [6] is one of the proposed language that attempts to unify concepts from some of these query languages, resulting in a new language that exploits the full versatility of XML. Its proposal has been adopted latter as the basis for the development of XQuery [12]. XQuery is intended primarily as a query language for querying collections of XML documents or documents viewed as XML (e.g., a SOAP representation of any data source). A look at the specification of XQuery reveals that the language is designed to be broadly capable of dealing with many sources of XML and not only query them, but also process them, yielding new XML structures as results. Existing implementations of XQuery such as Galax [20], XQEngine [33], IPSI-XQ [22], etc. do not

implement all the proposed functionality of the still-evolving specification. The Java Community Process (JCP) XQuery API for Java (XQJ) [34] is being designed as a common API for XQuery and is based on the XQuery Data Model [13] rather than a relational model. It is meant to be used for accessing data sources, much like JDBC does for SQL queries.

Based on our needs, XQuery would be a bulky language that has a lot of functionality that we don't need and does not address some of our requirements such as dealing with classification schemes and type of objects stored in the MathBroker registry. Instead, we exploited some of its features and its predecessors features in the design of MSQL query language. For instance MSQL borrows from XQuery [12] respectively XPath [11], QUILT [6], and their predecessors the syntax for navigating in the hierarchical structure of MSDL documents. From SQL [1] MSQL exploited the idea of a series of clauses based on keywords that provide a query pattern (the SELECT-FROM-WHERE pattern in SQL). Also from XQuery and its predecessors MSQL exploited the notion of a functional language composed of different kinds of expressions that can be nested.

Considering mathematical-oriented tools, there is a number of approaches [15] for searching and retrieving mathematical content, they range from basic textual methods to semantic-driven techniques. Their functionality a big deal depends on the structure, representation as well as storability of the target mathematical objects/documents. Most of them, for example, do not consider XML as a representation of mathematical objects. One approach [26] for achieving service discovery performs matchmaking between representations of tasks (client requests) and capabilities (service descriptions). It is performed only at the level of input, output, pre- and post- conditions which is not sufficient for full service discovery considering the richness of service descriptions.

At the semantic level of information representation and searchability which build on the foundation of XML, there exists a range of languages such as RDF [10] and OWL [9] and a number of Query tools for each kind of representation (see [2] for a review on these tools). These tools are under comprehensive development and up to this time there is no consensus on common ground (like in XQuery) for querying semantic content.

Several query languages have their semantics defined formally. The formal semantics of XQuery and XPath [14] provide a processing model as well as a description of static and dynamic semantics of the language. The formal specification uses operational semantics which is specified using value inference rules. Inference rules relate XPath/XQuery expressions to values and specify the order in which an expression is evaluated.

Aside from this W3C working draft, there are several attempts to formalize the semantics of XQuery and use them for implementation. In [7] an architecture with its implementation is presented based on the formal semantics of the language. It described some logical and physical optimizations based on the language formal semantics. In [21] a formal definition of a sublanguage of XQuery is given ignoring typing, namespaces, comments, programming instructions, and entities. It concentrates on certain aspects of the language that make it simple and expressive such as the role of expressions and XPath in the language. In [32] and [31] a similar approach for XPath and XSLT is employed using denotational semantics. In [19] the semantics of XQuery core is defined as static semantics and dynamic semantics. Static semantics is defined by means of type inference rules and dynamic semantics is defined by means of value inference rules. They both are used to map core expressions to operations on the XQuery Data Model.

7 Conclusion

In this report, we presented the syntax, semantics, architecture, and implementation of the Mathematical Services Query Language (MSQL). MSQL is a light-wight, content-based, functional query language specifically developed for querying the contents of mathematical documents formed in Mathematical Services Description Language (MSDL) where these documents are published in the MathBroker registry. MSQL complements the metadata-based querying facility of the MathBroker registry which facilitates querying of metadata accompanying descriptions published in the registry.

MSDL is rich of semantic content, and we are working to extend MSQL to accommodate semantic based query expressions that would require true reasoning. The query engine will contact a reasoner to reason about certain query expressions and return the corresponding decisions to the evaluator which uses these in processing MSQL queries.

A MSQL Grammar in ANTLR Syntax

```
header {
    package at.ac.uni_linz.risc.mathbroker.msql.msqlParser;
    import *;
}
class MsqlParser extends Parser;
options {
    exportVocab=Msql;                // Call the vocabulary "msql"
    buildAST = true;                 // build the Abstract Syntax Tree
}
tokens
{
 SELECT = "select" <AST=SelectNode>;
 FROM = "from" <AST=FromNode>;
 RETURN = "return" <AST=ReturnNode>;
 WHERE = "where" <AST=WhereNode>;
 SERVICE = "service" <AST=ServiceNode>;
 IMPLEMENTATION = "implementation" <AST=ImplementationNode>;
 ALGORITHM = "algorithm" <AST=AlgorithmNode>;
 PROBLEM = "problem" <AST=ProblemNode>;
 MACHINE = "machine" <AST=MachineNode>;
 COUNT = "count" <AST=CountNode>;
 CONTAINS = "contains" <AST=ContainsNode>;
 EMPTY = "empty" <AST=EmptyNode>;
 POSITION = "position" <AST=PositionNode>;
 DOC = "doc" <AST=DocNode>;
 EXISTS = "exists" <AST=ExistsNode>;
 ORDERBY = "orderby"<AST=OrderbyNode>;
 ASCENDING = "ascending" <AST=AscendingNode>;
 DESCENDING = "descending" <AST=DescendingNode>;
 SOME = "some" <AST=SomeNode>;
 EVERY = "every" <AST=EveryNode>;
 IN = "in" <AST=InNode>;
 IF = "if" <AST=IfExpr>;
 THEN = "then" <AST=ThenNode>;
 ELSE = "else" <AST=ElseNode>;
 LET = "let" <AST=LetExpr>;
 SATISFIES = "satisfies" <AST=SatisfiesNode>;
 AND = "and" <AST=AndExpr>;
 OR = "or" <AST=OrExpr>;
 NOT = "not"<AST=NotExpr>;
 RPATHEXPR;
```

```
 ARITHMETICEXPR;
 APATHEXPR;
 STEPEXPR;
 REGULAREXPR;
 STEP;
 EQUALS<AST=EqualsNode> ;
 NE<AST=NENode> ;
 LT<AST=LTNode> ;
 LTE<AST=LTENode> ;
 GT<AST=GTNode> ;
 GTE<AST=GTENode> ;
 PLUS<AST=PlusExpr> ;
 MINUS<AST=MinusExpr> ;
 STAR<AST=MultExpr> ;
 SLASH<AST=SlashNode> ;
 SLASHSLASH<AST=SlashSlashNode> ;
 }

msqlQuery : SELECT^ (EVERY | SOME) msdlEntity (FROM classification)?
    (whereClause)? (orderbyExpression)?
    ;
msdlEntity : SERVICE | IMPLEMENTATION | ALGORITHM | PROBLEM | MACHINE
    ;
classification : (SLASH<AST=ClassificationSlashNode>
      (LETTER<AST=LetterNode>
        | "problem"<AST=ProblemStep>
        | "service"<AST=ServiceStep>
        | "algorithm"<AST=AlgorithmStep>
        | "implementation"<AST=ImplementationStep>
        | STRING<AST=StringNode> ))*
        ;

whereClause : WHERE^ msqlExpr
    ;
msqlExpr : (notExpr | quantifiedExpr | ifExpr | letExpr)
    ;
notExpr : (NOT^)? orExpr
    ;
orExpr : andExpr (OR^ andExpr)*
    ;
andExpr : comparisonExpr (AND^ comparisonExpr)*
    ;
comparisonExpr : arithmeticExpr ( (EQUALS^ | NE^ | LT^ | LTE^ | GT^ | GTE^) arithmeticExpr)*
    ;
arithmeticExpr : additiveExpr ((PLUS^ | MINUS^) additiveExpr)*
    ;
additiveExpr : pathExpr ( STAR^ pathExpr)*
    ;
pathExpr : regularExpr {#pathExpr = #([RPATHEXPR, "pathExpr", "RelativePathExpr"], #pathExpr); }
        | stepDel regularExpr
         { #pathExpr = #([APATHEXPR, "pathExpr", "AbsolutePathExpr"], #pathExpr); }
         ;
regularExpr : stepExpr (stepDel stepExpr)*
    { #regularExpr = #([REGULAREXPR, "regularExpr", "RegularExpr"], #regularExpr); }
;
stepExpr : step (predicate)*
            { #stepExpr = #([STEP, "stepExpr", "Step"], #stepExpr); }
    | primaryExpr (predicate)* { #stepExpr = #([STEPEXPR, "stepExpr", "PrimaryExpr"], #stepExpr); }
    ;
primaryExpr : var
```

```
102      | literal
103      | msqlFunction
104      | DOT<AST=DotNode>
105      | LPAR^<AST=LPARNode> msqlExpr RPAR!
106      ;
107  var : DOLLAR^<AST=VarExpr> qName
108      ;
109  literal : numericLiteral | stringLiteral | booleanLiteral
110      ;
111  numericLiteral : NUMBER<AST=NumberNode>
112      ;
113  stringLiteral : STRING<AST=StringNode>
114      ;
115  booleanLiteral : "true"<AST=TrueNode>
116      | "false"<AST=FalseNode>
117      ;
118  stepDel : SLASH | SLASHSLASH
119      ;
120  step : nameTest | attributeStep
121      | "problem"<AST=ProblemStep>
122      | "service"<AST=ServiceStep>
123      | "algorithm"<AST=AlgorithmStep>
124      | "implementation"<AST=ImplementationStep>
125      ;
126  attributeStep : AT^<AST=AtNode> nameTest
127      ;
128  nameTest : LETTER<AST=LetterNode>
129      ;
130  qName: LETTER<AST=LetterNode>
131      ;
132  predicate : LB<AST=LBracketNode> msqlExpr
133      RB<AST=RBracketNode>
134      ;
135  orderbyExpression : ORDERBY^ msqlExpr
136      (COMMA<AST=CommaNode> msqlExpr)*
137          (sort)?
138          ;
139  sort : ASCENDING | DESCENDING
140      ;
141  msqlFunction : fname LPAR! (exprList)?
142      RPAR<AST=RPARNode>
143       ;
144  fname : COUNT | CONTAINS | EMPTY | POSITION | DOC | EXISTS
145      ;
146  exprList : msqlExpr
147      (COMMA<AST=CommaNode> msqlExpr)*
148          ;
149  quantifiedExpr : quantifier var IN<AST=InNode> msqlExpr
150       (COMMA<AST=CommaNode> var
151       IN<AST=InNode> msqlExpr)*
152       SATISFIES<AST=SatisfiesNode> msqlExpr
153       ;
154  quantifier : SOME^<AST=SomeQuantifiedExpr>
155      | EVERY^<AST=EveryQuantifiedExpr>
156      ;
157  ifExpr : IF^ msqlExpr THEN msqlExpr ELSE msqlExpr
158      ;
159  letExpr : LET^ letAssignmentClause
160      (COMMA<AST=CommaNode> letAssignmentClause)*
161          IN<AST=InNode> msqlExpr
```

```
        ;
letAssignmentClause : var ASSIGNMENT^<AST=AssignmentNode>
     msqlExpr
      ;

//================== Lexer =======================================

class MsqlLexer extends Lexer;

options {
    testLiterals=false;  //automatically test for literals
    k=4;
    charVocabulary='\3'..'\377' ; // allow ascii
    importVocab=Msql;
}

SLASH : '/' ;
SLASHSLASH : '/''/' ;
AT : '@';
DOLLAR : '$' ;
DOT : '.';
LB : '[';
RB : ']';
LPAR : '(';
RPAR : ')';
COMMA : ',';
COLON : ':';
ASSIGNMENT : ":=";
EQUALS : '=' ;
LT : '<' ;
LTE : "<=" ;
GT : '>' ;
GTE : ">=" ;
NE : "!=" ;
PLUS : '+' ;
MINUS : '-' ;
STAR : '*' ;
protected
DOUBLE_QUOTE_STRING  : '"'! (~('"'))* '"'! ;
STRING: DOUBLE_QUOTE_STRING ;
protected
DIGIT : ('0'..'9') ;
LETTER options { testLiterals=true;} : ('a'..'z'|'A'..'Z'|'_')
                                       ('a'..'z'|'A'..'Z'|'-'|'_'|'0'..'9'|'.'|':')* ;
NUMBER : (DIGIT)+ (DOT (DIGIT)+)? ;
WS : ( ' '
       // | '\r' '\n'
          | '\r'
        | '\n'
        | '\t'
        )+
        {$setType(Token.SKIP);}
      ;
EXIT : ';' { System.out.println(); System.exit(0);};
```

B A Client Application Using MSQL API

```
import java.io.*;
import java.util.*;
```

```
3  import at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST;
4  import at.ac.uni_linz.risc.mathbroker.registry.infomodel.*;
5  import javax.xml.bind.JAXBException;
6  import javax.xml.parsers.DocumentBuilder;
7  import javax.xml.parsers.DocumentBuilderFactory;
8  import javax.xml.registry.JAXRException;
9  import org.w3c.dom.Document;
10 import org.w3c.dom.Element;
11 import org.w3c.dom.NodeList;
12 import org.w3c.dom.Text;
13
14 /**
15  * This class demonstrates the usage of the Mathematical Services Query Language (MSQL) API.
16  * It uses the API in the follwoing manner:
17  * It reads a set of MSQL queries from an xml file,
18  * Executes each query against the MSDL documents fetched from the registry.
19  * Displays those documents satisfying each query.
20  *
21  * @author Rebhi Baraka
22  */
23 public class MsqlApiDemo {
24
25     public static void main(String[] args){
26         MsqlApiDemo msqlApiDemo = new MsqlApiDemo();
27         String registryURL = "http://koyote.risc.uni-linz.ac.at:8080/omar/registry/soap";
28         Properties connectionProps = new Properties();
29         MsqlQueryEvaluator msqlQueryEvaluatorImpl = new MsqlQueryEvaluatorImpl();
30         MathBrokerConnection registryConnection;
31         String userInput;
32         System.out.println("MSQL API demo");
33         try {
34             FileInputStream fileInputStream = new FileInputStream("documents/case.xml");
35             InputStream queryInputStream = fileInputStream;
36             MsqlQuery msqlBase = new MsqlQueryImpl();
37             //make connection to the MathBroker registry.
38              System.out.println("Connecting to MathBroker registry... \n");
39              connectionProps.put("javax.xml.registry.queryManagerURL", registryURL);
40              registryConnection = msqlBase.makeConnection(connectionProps);
41             //Used to build a dom model for the xml file containing the query.
42             DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
43             factory.setNamespaceAware(true);
44             DocumentBuilder builder = factory.newDocumentBuilder();
45             //xml document containing MSQL use cases.
46             Document queryDoc = (Document)builder.parse(queryInputStream);
47             NodeList cases = queryDoc.getElementsByTagName("query");
48             for (int i = 0; i < cases.getLength(); i++){
49                 Element element = (Element)cases.item(i);
50                 Text text = (Text)element.getFirstChild();
51                 String queryString = text.getData();
52                 System.out.println();
53                 System.out.println("Parsing the Query... \n");
54                 ChildAST queryTree = msqlBase.parseQuery(queryString);
55                 msqlBase.printQuery(queryTree);
56                 System.out.println("Retreiving MSDL documents from the registry
57                                     and performing the query on them... \n");
58                 Collection resultsCollection = msqlBase.performQuery(queryTree, registryConnection);
59                 System.out.println("Printing MSDL documents satisfying the query...");
60                 msqlApiDemo.resultIterator(resultsCollection);
61                 System.out.println("************ End of results for this query ***********");
62             }
```

```
63          }catch (Exception e) {
64             e.printStackTrace();
65          }
66      }
67
68      /**
69       * This method simply iterates over the documents satisfying the query
70       * and prints them according to the user preference;
71       * either one by one or all at once.
72       * @param results The collection of satisfying documents to be printed out.
73       */
74      public void resultIterator(Collection results) throws IOException,
75       JAXRException, JAXBException {
76          BufferedReader input = new BufferedReader(new InputStreamReader(System.in));
77          String userInput="";
78          System.out.println("  press ENTER to see next resulting document or type 'all'
79                             to print out all resulting documents for the current query");
80          Iterator resultIter = results.iterator();
81          try {
82          userInput = input.readLine();
83          if (userInput.equalsIgnoreCase("this"))
84              while (resultIter.hasNext()){
85                  Object obj = resultIter.next();
86                  if (obj instanceof MathBrokerObject){
87                      MathBrokerObject mathBrokerObject = (MathBrokerObject)obj;
88
89                      System.out.println("-----------------------------------------");
90                      mathBrokerObject.print();
91                  }
92              }
93          else if (userInput.equalsIgnoreCase("all"))
94              while (resultIter.hasNext()){
95                      MathBrokerObject mathBrokerObject = (MathBrokerObject)resultIter.next();
96                      System.out.println("-----------------------------------------");
97                      mathBrokerObject.print();
98              }
99          else
100             while (resultIter.hasNext()){
101                 System.out.println("-----------------------------------------------");
102                 MathBrokerObject mathBrokerObject = (MathBrokerObject)resultIter.next();
103                 mathBrokerObject.print();
104                 userInput = input.readLine();
105             }
106              }catch (Exception e) {
107         System.err.println();
108            }
109     }
110 }
```

C A Sample MSDL Service Description

```
1  <monet:definitions
2   targetnamespace="http://risc.uni-linz.ac.at/mathbroker/RischIndefIntegration"
3   xmlns:dc="http://purl.org/dc/elements/1.1/"
4   xmlns:mathb="http://risc.uni-linz.ac.at/mathbroker/ns"
5   xmlns:monet="http://monet.nag.co.uk/monet/OpenMathDC"
6   xmlns:om="http://www.openmath.org/OpenMath"
7   xmlns:symbint="http://perseus.risc.uni-linz.ac.at:8080/axis/services/SymbolicIntegration"
8   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
```

```
 9  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
10        xsi:schemalocation="http://monet.nag.co.uk/monet/OpenMathDC/
11                  home/olga/cvs/perseus/monet-based-xsd/xsd/monetOM_DC.xsd">
12   <!--  $Id: risch.xml,v 1.1 2004/04/23 10:49:52 rbaraka Exp $ -->
13   <!-- actually these are just directives - at this stage no tool processes these -->
14   <mathb:machine_hardware address="193.170.37.69" name="perseus.risc.uni-linz.ac.at">
15     <mathb:CPU name="Intel Celeron"></mathb:CPU>
16     <mathb:CPU_speed mhz="733"></mathb:CPU_speed>
17     <mathb:RAMsize mb="256"></mathb:RAMsize>
18     <mathb:disksize gb="40"></mathb:disksize>
19     <mathb:OS href="http://www.suse.de"></mathb:OS>
20   </mathb:machine_hardware>
21   <monet:import location="./SymbolicIntegration.wsdl"
22     namespace="http://perseus.risc.uni-linz.ac.at:8080/axis/services/SymbolicIntegration">
23   </monet:import>
24   <monet:problem name="indefinite-integration">
25     <monet:header></monet:header>
26     <monet:body>
27       <monet:input name="f">
28         <monet:signature>
29           <om:OMOBJ>
30             <om:OMA>
31               <om:OMS cd="sts" name="mapsto"></om:OMS>
32               <om:OMS cd="setname1" name="R"></om:OMS>
33               <om:OMS cd="setname1" name="R"></om:OMS>
34               <om:OMS cd="setname1" name="R"></om:OMS>
35             </om:OMA>
36           </om:OMOBJ>
37         </monet:signature>
38       </monet:input>
39       <monet:output name="i">
40         <monet:signature>
41           <om:OMOBJ>
42             <om:OMA>
43               <om:OMS cd="sts" name="mapsto"></om:OMS>
44               <om:OMS cd="setname1" name="R"></om:OMS>
45               <om:OMS cd="setname1" name="R"></om:OMS>
46               <om:OMS cd="setname1" name="R"></om:OMS>
47             </om:OMA>
48           </om:OMOBJ>
49         </monet:signature>
50       </monet:output>
51       <monet:post-condition>
52         <om:OMOBJ>
53           <om:OMA>
54             <om:OMS cd="relation1" name="eq"></om:OMS>
55             <om:OMV name="i"></om:OMV>
56             <om:OMA>
57               <om:OMS cd="calculus1" name="indefint"></om:OMS>
58               <om:OMV name="f"></om:OMV>
59             </om:OMA>
60           </om:OMA>
61         </om:OMOBJ>
62       </monet:post-condition>
63     </monet:body>
64   </monet:problem>
65   <monet:algorithm name="RischAlg">
66     <monet:documentation>This is the metadata for the algorithm
67       Risch. The namespace is the target namespace of this document.
68       </monet:documentation>
```

```
69      <monet:bibliography href="http://www.emis.de/cgi-bin/zmen/ZMATH/en/
70                                      quick.html?type=xml&an=0184.06702">
71        <!-- more dublin core -->
72        <monet:documentation> Dublin Core Data </monet:documentation>
73        <dc:creator>Risch,R.H.</dc:creator>
74        <dc:title>The Problem of Integration in Finite Terms</dc:title>
75        <dc:source> Trans. A.M.S. 139 pp.167 - 189</dc:source>
76        <dc:publisher>AMS</dc:publisher>
77        <dc:date>1969</dc:date>
78      </monet:bibliography>
79    </monet:algorithm>
80      <monet:implementation name="RImpl">
81      <mathb:efficiency_factor wrt="S200Spec">
82        <mathb:speed>1.1</mathb:speed>
83        <mathb:throughput>0.7</mathb:throughput>
84      </mathb:efficiency_factor>
85        <monet:software href="http://www.wolfram.com"></monet:software>
86        <monet:software href="http://riaca.win.tue.nl/software/ROML">
87        </monet:software>
88        <monet:hardware href="http://risc.uni-linz.ac.at/mathbroker/
89                          RischIndefIntegration/perseus.risc.uni-linz.ac.at">
90        </monet:hardware>
91        <monet:algorithm href="http://risc.uni-linz.ac.at/
92                          mathbroker/RischIndefIntegration/RischAlg">
93        </monet:algorithm>
94      </monet:implementation>
95    <monet:service name="RRISC">
96        <monet:documentation>This is an implementation of the algorithm
97          Risch. We use the mathb namespace to state expected performance
98          of the  concrete implementation wrt to its theoretical
99          complexity measure.</monet:documentation>
100     <monet:classification>
101       <monet:problem href="http://risc.uni-linz.ac.at/mathbroker/
102                         RischIndefIntegration/indefinite-integration">
103       </monet:problem>
104     </monet:classification>
105     <monet:implementation href="http://risc.uni-linz.ac.at/mathbroker/
106                           RischIndefIntegration/RImpl">
107     </monet:implementation>
108     <monet:service-interface-description
109       href="http://perseus.risc.uni-linz.ac.at:8080/axis/
110              services/SymbolicIntegration?wsdl">
111     </monet:service-interface-description>
112     <monet:service-binding>
113       <monet:map action="exec" operation="symbint:Integrator:indefInt"
114        problem-reference="indefinite-integration"></monet:map>
115       <monet:message-construction io-ref="f"
116       message-name="symbint:IndefIntRequest" message-part="in0">
117       </monet:message-construction>
118     </monet:service-binding>
119     <monet:service-metadata></monet:service-metadata>
120     <monet:broker-interface>
121       <monet:service-URI></monet:service-URI>
122     </monet:broker-interface>
123   </monet:service>
124 </monet:definitions>
```

D MSQL API

D.1 Package at.ac.uni_linz.risc.mathbroker.msql.msqlParser

Interface MsqlLexerTokenTypes

Declaration public interface MsqlLexerTokenTypes

All known subinterfaces MsqlLexer

All classes known to implement interface MsqlLexer

Interface MsqlTokenTypes

Declaration public interface MsqlTokenTypes

All known subinterfaces MsqlParser

All classes known to implement interface MsqlParser

Class MsqlLexer

Declaration public class MsqlLexer
extends antlr.CharScanner antlr.CharScanner
implements MsqlLexerTokenTypes, antlr.TokenStream

Constructors

- `public` **MsqlLexer**`( antlr.InputBuffer` **ib** `)` MsqlLexer
- `public` **MsqlLexer**`( java.io.InputStream` **in** `)` MsqlLexer
- `public` **MsqlLexer**`( antlr.LexerSharedInputState` **state** `)` MsqlLexer
- `public` **MsqlLexer**`( java.io.Reader` **in** `)`

Methods

- `public final void` **mASSIGNMENT**`( boolean` **_createToken** `) throws antlr.RecognitionException, antlr.CharStreamException, antlr.TokenStreamException` mAT
- `public final void` **mAT**`( boolean` **_createToken** `) throws antlr.RecognitionException, antlr.CharStreamException, antlr.TokenStreamException` mCOLON
- `public final void` **mCOLON**`( boolean` **_createToken** `) throws antlr.RecognitionException, antlr.CharStreamException, antlr.TokenStreamException` mCOMMA
- `public final void` **mCOMMA**`( boolean` **_createToken** `) throws antlr.RecognitionException, antlr.CharStreamException, antlr.TokenStreamException` mDIGIT

- protected final void **mDIGIT**(boolean **_createToken**) throws antlr.RecognitionException, antlr.CharStreamException, antlr.TokenStreamException mDOLLAR
- public final void **mDOLLAR**(boolean **_createToken**) throws antlr.RecognitionException, antlr.CharStreamException, antlr.TokenStreamException mDOT
- public final void **mDOT**(boolean **_createToken**) throws antlr.RecognitionException, antlr.CharStreamException, antlr.TokenStreamException mDOUBLE_QUOTE_STRING
- protected final void **mDOUBLE_QUOTE_STRING**(boolean **_createToken**) throws antlr.RecognitionException, antlr.CharStreamException, antlr.TokenStreamException mEQUALS
- public final void **mEQUALS**(boolean **_createToken**) throws antlr.RecognitionException, antlr.CharStreamException, antlr.TokenStreamException mEXIT
- public final void **mEXIT**(boolean **_createToken**) throws antlr.RecognitionException, antlr.CharStreamException, antlr.TokenStreamException mGT
- public final void **mGT**(boolean **_createToken**) throws antlr.RecognitionException, antlr.CharStreamException, antlr.TokenStreamException mGTE
- public final void **mGTE**(boolean **_createToken**) throws antlr.RecognitionException, antlr.CharStreamException, antlr.TokenStreamException mLB
- public final void **mLB**(boolean **_createToken**) throws antlr.RecognitionException, antlr.CharStreamException, antlr.TokenStreamException mLETTER
- public final void **mLETTER**(boolean **_createToken**) throws antlr.RecognitionException, antlr.CharStreamException, antlr.TokenStreamException mLPAR
- public final void **mLPAR**(boolean **_createToken**) throws antlr.RecognitionException, antlr.CharStreamException, antlr.TokenStreamException mLT
- public final void **mLT**(boolean **_createToken**) throws antlr.RecognitionException, antlr.CharStreamException, antlr.TokenStreamException mLTE
- public final void **mLTE**(boolean **_createToken**) throws antlr.RecognitionException, antlr.CharStreamException, antlr.TokenStreamException mMINUS
- public final void **mMINUS**(boolean **_createToken**) throws antlr.RecognitionException, antlr.CharStreamException, antlr.TokenStreamException mNE
- public final void **mNE**(boolean **_createToken**) throws antlr.RecognitionException, antlr.CharStreamException, antlr.TokenStreamException mNUMBER
- public final void **mNUMBER**(boolean **_createToken**) throws antlr.RecognitionException, antlr.CharStreamException, antlr.TokenStreamException mPLUS

- `public final void` **mPLUS**`( boolean` **_createToken** `) throws antlr.RecognitionException, antlr.CharStreamException, antlr.TokenStreamException` mRB
- `public final void` **mRB**`( boolean` **_createToken** `) throws antlr.RecognitionException, antlr.CharStreamException, antlr.TokenStreamException` mRPAR
- `public final void` **mRPAR**`( boolean` **_createToken** `) throws antlr.RecognitionException, antlr.CharStreamException, antlr.TokenStreamException` mSLASH
- `public final void` **mSLASH**`( boolean` **_createToken** `) throws antlr.RecognitionException, antlr.CharStreamException, antlr.TokenStreamException` mSLASHSLASH
- `public final void` **mSLASHSLASH**`( boolean` **_createToken** `) throws antlr.RecognitionException, antlr.CharStreamException, antlr.TokenStreamException` mSTAR
- `public final void` **mSTAR**`( boolean` **_createToken** `) throws antlr.RecognitionException, antlr.CharStreamException, antlr.TokenStreamException` mSTRING
- `public final void` **mSTRING**`( boolean` **_createToken** `) throws antlr.RecognitionException, antlr.CharStreamException, antlr.TokenStreamException` mWS
- `public final void` **mWS**`( boolean` **_createToken** `) throws antlr.RecognitionException, antlr.CharStreamException, antlr.TokenStreamException` nextToken
- `antlr.Token` **nextToken**`( ) throws antlr.TokenStreamException`

Class MsqlParser

Declaration public class MsqlParser
extends antlr.LLkParser antlr.LLkParser
implements MsqlTokenTypes

Constructors

- `public` **MsqlParser**`( antlr.ParserSharedInputState` **state** `)` MsqlParser
- `public` **MsqlParser**`( antlr.TokenBuffer` **tokenBuf** `)` MsqlParser
- `protected` **MsqlParser**`( antlr.TokenBuffer` **tokenBuf**`, int` **k** `)` MsqlParser
- `public` **MsqlParser**`( antlr.TokenStream` **lexer** `)` MsqlParser
- `protected` **MsqlParser**`( antlr.TokenStream` **lexer**`, int` **k** `)`

Methods

- `public final void` **additiveExpr**`( ) throws antlr.RecognitionException, antlr.TokenStreamException` andExpr
- `public final void` **andExpr**`( ) throws antlr.RecognitionException, antlr.TokenStreamException` arithmeticExpr
- `public final void` **arithmeticExpr**`( ) throws antlr.RecognitionException, antlr.TokenStreamException` attributeStep

- `public final void` **attributeStep**`( ) throws antlr.RecognitionException, antlr.TokenStreamException` booleanLiteral
- `public final void` **booleanLiteral**`( ) throws antlr.RecognitionException, antlr.TokenStreamException` buildTokenTypeASTClassMap
- `protected void` **buildTokenTypeASTClassMap**`( )` classification
- `public final void` **classification**`( ) throws antlr.RecognitionException, antlr.TokenStreamException` comparisonExpr
- `public final void` **comparisonExpr**`( ) throws antlr.RecognitionException, antlr.TokenStreamException` exprList
- `public final void` **exprList**`( ) throws antlr.RecognitionException, antlr.TokenStreamException` fname
- `public final void` **fname**`( ) throws antlr.RecognitionException, antlr.TokenStreamException` ifExpr
- `public final void` **ifExpr**`( ) throws antlr.RecognitionException, antlr.TokenStreamException` letAssignmentClause
- `public final void` **letAssignmentClause**`( ) throws antlr.RecognitionException, antlr.TokenStreamException` letExpr
- `public final void` **letExpr**`( ) throws antlr.RecognitionException, antlr.TokenStreamException` literal
- `public final void` **literal**`( ) throws antlr.RecognitionException, antlr.TokenStreamException` msdlEntity
- `public final void` **msdlEntity**`( ) throws antlr.RecognitionException, antlr.TokenStreamException` msqlExpr
- `public final void` **msqlExpr**`( ) throws antlr.RecognitionException, antlr.TokenStreamException` msqlFunction
- `public final void` **msqlFunction**`( ) throws antlr.RecognitionException, antlr.TokenStreamException` msqlQuery
- `public final void` **msqlQuery**`( ) throws antlr.RecognitionException, antlr.TokenStreamException` nameTest
- `public final void` **nameTest**`( ) throws antlr.RecognitionException, antlr.TokenStreamException` notExpr
- `public final void` **notExpr**`( ) throws antlr.RecognitionException, antlr.TokenStreamException` numericLiteral
- `public final void` **numericLiteral**`( ) throws antlr.RecognitionException, antlr.TokenStreamException` orderbyExpression
- `public final void` **orderbyExpression**`( ) throws antlr.RecognitionException, antlr.TokenStreamException` orExpr
- `public final void` **orExpr**`( ) throws antlr.RecognitionException, antlr.TokenStreamException` pathExpr
- `public final void` **pathExpr**`( ) throws antlr.RecognitionException, antlr.TokenStreamException` predicate
- `public final void` **predicate**`( ) throws antlr.RecognitionException, antlr.TokenStreamException` primaryExpr
- `public final void` **primaryExpr**`( ) throws antlr.RecognitionException, antlr.TokenStreamException` qName

- `public final void` **qName**`( ) throws antlr.RecognitionException, antlr.TokenStreamException` quantifiedExpr
- `public final void` **quantifiedExpr**`( ) throws antlr.RecognitionException, antlr.TokenStreamException` quantifier
- `public final void` **quantifier**`( ) throws antlr.RecognitionException, antlr.TokenStreamException` regularExpr
- `public final void` **regularExpr**`( ) throws antlr.RecognitionException, antlr.TokenStreamException` sort
- `public final void` **sort**`( ) throws antlr.RecognitionException, antlr.TokenStreamException` step
- `public final void` **step**`( ) throws antlr.RecognitionException, antlr.TokenStreamException` stepDel
- `public final void` **stepDel**`( ) throws antlr.RecognitionException, antlr.TokenStreamException` stepExpr
- `public final void` **stepExpr**`( ) throws antlr.RecognitionException, antlr.TokenStreamException` stringLiteral
- `public final void` **stringLiteral**`( ) throws antlr.RecognitionException, antlr.TokenStreamException` var
- `public final void` **var**`( ) throws antlr.RecognitionException, antlr.TokenStreamException` whereClause
- `public final void` **whereClause**`( ) throws antlr.RecognitionException, antlr.TokenStreamException`

D.2 Package at.ac.uni_linz.risc.mathbroker.msql.msqlSemantics

Interface ExprEvaluator

The interface to evaluate arithmetic, logical, comparative, quantified, if, and let MSQL expressions.

Declaration public interface ExprEvaluator

All known subinterfaces ExprEvaluatorImpl

All classes known to implement interface ExprEvaluatorImpl

Interface MsqlQuery

See also

- `MsqlApiDemoMsqlApiDemo` MsqlApiDemo
- `MsqlApiDemo2MsqlApiDemo2` MsqlApiDemo2

Declaration public interface MsqlQuery

All known subinterfaces MsqlQueryImpl

All classes known to implement interface MsqlQueryImpl

METHODS

- `java.util.Iterator` **iterateQuery**`(`
  `at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST` **msqlTree**`,`
  `MathBrokerConnection` **registryConnection** `) throws java.lang.Exception`
  - **Description**
    This method returns an iterator which performs the query against each candidate document. Unlike performQuery, it does not collect the resulting documents. This method obviosly does not allow sorting. It also does not support some quantifier bec. iterator is meant to be used with every quantifier.
  - **Parameters**
    - `msqlTree` – The parsed AST of the query string.
    - `registryConnection` – The connection to the registry where MSDL docs would be retrieved.
  - **Returns** – Iterator an iterator which performs the query against each candidate document.
  - **Throws**
    - `java.lang.Exception` –

  makeConnection

- `MathBrokerConnection` **makeConnection**`( java.util.Properties` **connectionProps** `) throws JAXRException`
  - **Description**
    This method creates the connection to Mathbroker registry.
  - **Parameters**
    - `connectionProps` – The url of the registry. By default is is the MathBroker registry. and the kind of connection which is to the query manager not for publishing.
  - **Throws**
    - `JAXRException` – If the connection properties are not defined.

  parseQuery

- `at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST` **parseQuery**`(`
  `java.lang.String` **queryString** `) throws java.lang.Exception`
  - **Description**
    This method parses the given query according to the MSQL parser rules.
  - **Parameters**
    - `queryString` – The query string.
  - **Returns** – The parsed AST tree of the given query.
  - **Throws**
    - `java.lang.Exception` – If the query string violates the parser rules.

  performQuery

- `java.util.Collection` **performQuery**`(`
  `at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST` **msqlTree**`,`
  `MathBrokerConnection` **registryConnection** `) throws java.lang.Exception`
  - **Description**
    This method performs the given query against the candidate MSDL documents retrieved from the registry. Candidate documents are determined according to the type of entity and classification concept of the query, e.g., Select every entityType form classificationConcept where ... It returns the resulting documents as a collection.

- **Parameters**
  - `msqlTree` – The parsed AST of the query string.
  - `registryConnection` – The connection to the registry where MSDL docs would be retrieved.
- **Returns** – The collection of MSDL documents satisfying the query.
- **Throws**
  - `java.lang.Exception` –

printQuery

- `void` **printQuery**`( at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST` **msqlTree** `)`
  - **Description**
    This method prints out the parsed query.
  - **Parameters**
    - `msqlTree` – The AST of the query to be printed.

Interface MsqlQueryEvaluator

Declaration public interface MsqlQueryEvaluator

All known subinterfaces MsqlQueryEvaluatorImpl

All classes known to implement interface MsqlQueryEvaluatorImpl

Methods

- `java.lang.String` **getClassificationName**`(`
  `at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST` **msqlQuery** `)`
  - **Description**
    Returns the string equivalent to the classification concept in the MathBroker registry. The default classification scheme is Mathbroker. It is assumed when the classification is not stated in the query.
  - **Parameters**
    - `msqlQuery` – The AST of the query.
  - **Returns** – The string equivalent to the classification concept.

getEntityName

- `java.lang.String` **getEntityName**`(`
  `at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST` **msqlQuery** `)`
  - **Description**
    Returns the string value equivalent to the enitity stated in the query. According to the MathBroker information model, entitities are problem, algorithm, implementation, service, and machine.
  - **Parameters**
    - `msqlQuery` – The AST of the query.
  - **Returns** – The string value equivalent to the enitity.

getOrderby

- `at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST` **getOrderby(** `at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST` **msqlQuery)**
  - **Description**
    Returns the MSQL expr after the orderby
  - **Parameters**
    * `msqlQuery` – The query AST.
  - **Returns** – The expression after orderby.

  getSomeOrEvery
- `java.lang.String` **getSomeOrEvery(** `at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST` **msqlQuery)**
  - **Description**
    Returns the string "some" or the string "every" after select.
  - **Parameters**
    * `msqlQuery` – The AST of the query from which the some or every can be found.
  - **Returns** – The string "some" or the string "every".

  getSort
- `java.lang.String` **getSort(** `at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST` **msqlQuery)**
  - **Description**
    Returns the string "ascending" or the string "descending" after the orderby expression.
  - **Parameters**
    * `msqlQuery` – The query AST.
  - **Returns** – The string "ascending" or the string "descending" after the orderby expression.

Interface PathExprEvaluator

Declaration public interface PathExprEvaluator

All known subinterfaces PathExprEvaluatorImpl

All classes known to implement interface PathExprEvaluatorImpl

Interface PrimaryExprEvaluator

Declaration public interface PrimaryExprEvaluator

All known subinterfaces PrimaryExprEvaluatorImpl

All classes known to implement interface PrimaryExprEvaluatorImpl

Interface RegularExprEvaluator

Declaration public interface RegularExprEvaluator

All known subinterfaces RegularExprEvaluatorImpl

All classes known to implement interface RegularExprEvaluatorImpl

Class BooleanValue

Represents a boolean value.

Declaration public class BooleanValue
extends at.ac.uni_linz.risc.mathbroker.msql.msqlSemantics.Value Value

Constructors

- `public` **BooleanValue**`( )` BooleanValue
- `public` **BooleanValue**`( boolean` **value** `)`
  - **Parameters**
    - `value` –

  BooleanValue
- `public` **BooleanValue**`( java.lang.String` **sVal** `)` BooleanValue
- `public` **BooleanValue**`( Value` **val** `)`

Methods

- `public boolean` **getValue**`( )`
  - **Returns** – Returns the value.

  setValue
- `public void` **setValue**`( boolean` **value** `)`
  - **Parameters**
    - `value` – The value to set.

Class Declaration

Has the declarations that are forwarded or returned by individual methods of each semantic function. For example: E [[V]] d n = d([[V]]) return the declaration d. While E [[P]] d n = P [[P]] d n forwards declaration d.

Declaration public class Declaration
extends java.lang.Object java.lang.Object

Constructors

- `public` **Declaration**`( )`

Methods

- `public Value` **get**`( java.lang.String` **key** `)`
  - **Description**
    Retrieves a variable declaration from the declaration hashtable.
  - **Parameters**
    - `key` – The variable name.
  - **Returns** – value The variable value.

  lookup
- `public Value` **lookup**`( java.lang.String` **varName** `)` lookup
- `public Value` **lookup**`( at.ac.uni_linz.risc.mathbroker.msql.treeWalker.VarExpr` **var** `)`
  - **Description**
    Searches the table for a variable
  - **Parameters**
    - `var` – The variable.
  - **Returns** – value The variable value.

  put
- `public void` **put**`( java.lang.String` **key**`, Value` **value** `)`
  - **Description**
    Stores a variable declaration in the declaration hashtable.
  - **Parameters**
    - `key` – The variable name.
    - `value` – The variable value.

  put
- `public void` **put**`( at.ac.uni_linz.risc.mathbroker.msql.treeWalker.VarExpr` **var**`, Value` **value** `)` update
- `public Declaration` **update**`( at.ac.uni_linz.risc.mathbroker.msql.treeWalker.VarExpr` **var**`, Value` **value** `) throws java.lang.CloneNotSupportedException`
  - **Description**
    Updates a variable declaration.
  - **Parameters**
    - `var` – The variable.
    - `value` – The new value of the variable.
  - **Returns** – declaration The declaration with the disgnated variable updated.
  - **Throws**
    - `java.lang.CloneNotSupportedException` –

Class DescendingComparator

Reverses the order of the given Comparator.

Declaration public class DescendingComparator
extends java.lang.Object java.lang.Object
implements java.util.Comparator

Constructors

- `public` **DescendingComparator**`( java.util.Comparator` **comparatorToReverse** `)`

Methods

- `public int` **compare**`( java.lang.Object` **obj1**`, java.lang.Object` **obj2** `)`
  - **Description**
    Returns a negative integer, zero, or a positive integer if the first argument is greater than, equal to, or less than the second, respectively.

Class ExprEvaluatorImpl

This class implements methods to evaluate arithmetic, logical, comparative, quantified, if, and let MSQL expressions. It uses the @link evaluateExpr method to decide which specific evaluate method should be called depending on the given MSQL expression AST.

Declaration public class ExprEvaluatorImpl
extends java.lang.Object java.lang.Object
implements ExprEvaluator

Constructors

- `public` **ExprEvaluatorImpl**`( )`

Class FunctionEvaluator

This class evaluates MSQL function expressions

Declaration public class FunctionEvaluator
extends java.lang.Object java.lang.Object

Constructors

- `public` **FunctionEvaluator**`( )`

Class IntValue

Represents an integer value.

Declaration public class IntValue
extends at.ac.uni_linz.risc.mathbroker.msql.msqlSemantics.Value Value

Constructors

- `public` **IntValue**`( )` IntValue
- `public` **IntValue**`( int` **value** `)`
  - **Parameters**
    - `value` –

  IntValue
- `public` **IntValue**`( Value` **val** `)`
  - **Parameters**
    - `val` – to be removed

METHODS

- `public int` **getValue**`( )`
  - **Returns** – value The value to get.

  setValue
- `public void` **setValue**`( int value )`
  - **Parameters**
    - `value` – The value to set.

CLASS LISTVALUE

Represents a set of nodes values.

DECLARATION public class ListValue
extends at.ac.uni_linz.risc.mathbroker.msql.msqlSemantics.Value Value

CONSTRUCTORS

- `public` **ListValue**`( )` ListValue
- `public` **ListValue**`( java.util.List value )`
  - **Description**
    Takes a list as a parameter
  - **Parameters**
    - `value` –

  ListValue
- `public` **ListValue**`( Node node )`
  - **Description**
    Takes a Node as a parameter
  - **Parameters**
    - `node` –

METHODS

- `public ListValue` **add**`( ListValue val ) throws MsqlException`
  - **Description**
    Adds a new list to the present list and returns the generated list.
  - **Parameters**
    - `val` – of the list to be added.
  - **Returns** – The generated list.
  - **Throws**
    - `at.ac.uni_linz.risc.mathbroker.msql.msqlSemantics.MsqlException` –

  getValue
- `public java.util.List` **getValue**`( )`
  - **Description**
    Returns the list value of the ListValue;
  - **Returns** – The list value.

  setValue
- `public void` **setValue**`( Value val )`

Class MsqlApiDemo

This class demonstrates the usage of the Mathematical Services Query Language (MSQL) API. It uses the API in the follwoing manner: It reads a set of MSQL queries from an xml file, Executes each query against the MSDL documents fetched from the registry. Displays those documents satisfying each query.

Declaration public class MsqlApiDemo
extends java.lang.Object java.lang.Object

Constructors

- `public` **MsqlApiDemo**`( )`

Methods

- `public static void` **main**`( java.lang.String[]` **args** `)` resultIterator
- `public void` **resultIterator**`( java.util.Collection` **results** `) throws java.io.IOException, JAXRException, JAXBException`
  - **Description**
    This method simply iterates over the documents satisfying the query and prints them according to the user preference; either one by one or all at once.
  - **Parameters**
    - `results` – The collection of satisfying documents to be printed out.

Class MsqlApiDemo2

Declaration public class MsqlApiDemo2
extends java.lang.Object java.lang.Object

Constructors

- `public` **MsqlApiDemo2**`( )`

Methods

- `public static void` **main**`( java.lang.String[]` **args** `) throws MsqlException, java.lang.Exception`

Class MsqlQueryEvaluatorImpl

Implementation of the evaluation of MSQL query. The evaluation of MSQL query starts here.

Declaration public class MsqlQueryEvaluatorImpl
extends java.lang.Object java.lang.Object
implements MsqlQueryEvaluator

Constructors

- `public` **MsqlQueryEvaluatorImpl**`( )`

- `public static Value` **evaluate(** `at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST` **msqlQuery,** `Declaration` **declaration,** `Node` **node**) `throws java.io.IOException, JAXRException, JAXBException, MsqlException, java.lang.CloneNotSupportedException`
  getClassificationName
- `java.lang.String` **getClassificationName(** `at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST` **msqlQuery**)
  - **Description copied from MsqlQueryEvaluatorMsqlQueryEvaluatorMsqlQueryEvaluator**
    Returns the string equivalent to the classification concept in the MathBroker registry. The default classification scheme is Mathbroker. It is assumed when the classification is not stated in the query.
  - **Parameters**
    - `msqlQuery` – The AST of the query.
  - **Returns** – The string equivalent to the classification concept.

  getEntityName
- `java.lang.String` **getEntityName(** `at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST` **msqlQuery**)
  - **Description copied from MsqlQueryEvaluatorMsqlQueryEvaluatorMsqlQueryEvaluator**
    Returns the string value equivalent to the enitity stated in the query. According to the MathBroker information model, entitities are problem, algorithm, implementation, service, and machine.
  - **Parameters**
    - `msqlQuery` – The AST of the query.
  - **Returns** – The string value equivalent to the enitity.

  getOrderby
- `at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST` **getOrderby(** `at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST` **msqlQuery**)
  - **Description copied from MsqlQueryEvaluatorMsqlQueryEvaluatorMsqlQueryEvaluator**
    Returns the MSQL expr after the orderby
  - **Parameters**
    - `msqlQuery` – The query AST.
  - **Returns** – The expression after orderby.

  getSomeOrEvery
- `java.lang.String` **getSomeOrEvery(** `at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST` **msqlQuery**)
  - **Description copied from MsqlQueryEvaluatorMsqlQueryEvaluatorMsqlQueryEvaluator**
    Returns the string "some" or the string "every" after select.
  - **Parameters**
    - `msqlQuery` – The AST of the query from which the some or every can be found.
  - **Returns** – The string "some" or the string "every".

getSort

- `java.lang.String` **getSort**`( at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST` **msqlQuery** `)`
  - **Description copied from MsqlQueryEvaluatorMsqlQueryEvaluatorMsqlQueryEvaluator**
    Returns the string "ascending" or the string "descending" after the orderby expression.
  - **Parameters**
    - `msqlQuery` – The query AST.
  - **Returns** – The string "ascending" or the string "descending" after the orderby expression.

Class MsqlQueryImpl

Declaration public class MsqlQueryImpl
extends java.lang.Object java.lang.Object
implements MsqlQuery

Constructors

- `public` **MsqlQueryImpl**`( )`

Methods

- `java.util.Iterator` **iterateQuery**`(`
  `at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST` **msqlTree**`,`
  `MathBrokerConnection` **registryConnection** `) throws java.lang.Exception`
  - **Description copied from MsqlQueryMsqlQueryMsqlQuery**
    This method returns an iterator which performs the query against each candidate document. Unlike performQuery, it does not collect the resulting documents. This method obviosly does not allow sorting. It also does not support some quantifier bec. iterator is meant to be used with every quantifier.
  - **Parameters**
    - `msqlTree` – The parsed AST of the query string.
    - `registryConnection` – The connection to the registry where MSDL docs would be retrieved.
  - **Returns** – Iterator an iterator which performs the query against each candidate document.
  - **Throws**
    - `java.lang.Exception` –

  makeConnection

- `MathBrokerConnection` **makeConnection**`( java.util.Properties` **connectionProps** `) throws JAXRException`
  - **Description copied from MsqlQueryMsqlQueryMsqlQuery**
    This method creates the connection to Mathbroker registry.
  - **Parameters**
    - `connectionProps` – The url of the registry. By default is is the MathBroker registry. and the kind of connection which is to the query manager not for publishing.
  - **Throws**
    - `JAXRException` – If the connection properties are not defined.

parseQuery

- `at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST` **parseQuery(** `java.lang.String` **queryString**) `throws java.lang.Exception`
  - **Description copied from MsqlQueryMsqlQueryMsqlQuery**
    This method parses the given query according to the MSQL parser rules.
  - **Parameters**
    - `queryString` – The query string.
  - **Returns** – The parsed AST tree of the given query.
  - **Throws**
    - `java.lang.Exception` – If the query string violates the parser rules.

performQuery

- `java.util.Collection` **performQuery(** `at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST` **msqlTree,** `MathBrokerConnection` **registryConnection**) `throws java.lang.Exception`
  - **Description copied from MsqlQueryMsqlQueryMsqlQuery**
    This method performs the given query against the candidate MSDL documents retrieved from the registry. Candidate documents are determined according to the type of entity and classification concept of the query, e.g., Select every entityType form classificationConcept where ... It returns the resulting documents as a collection.
  - **Parameters**
    - `msqlTree` – The parsed AST of the query string.
    - `registryConnection` – The connection to the registry where MSDL docs would be retrieved.
  - **Returns** – The collection of MSDL documents satisfying the query.
  - **Throws**
    - `java.lang.Exception` –

printQuery

- `void` **printQuery(** `at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST` **msqlTree**)
  - **Description copied from MsqlQueryMsqlQueryMsqlQuery**
    This method prints out the parsed query.
  - **Parameters**
    - `msqlTree` – The AST of the query to be printed.

Class Node

This class represents an MSDL description node. It models the contents in DOM

Declaration public class Node
extends java.lang.Object java.lang.Object

Constructors

- `public` **Node**`( )`
  - **Description**
    Node constructor.

  Node

- `public` **Node**`( org.w3c.dom.Document` **doc** `)`
  - **Description**
    Node constructor with a DOM document as parameter.
  - **Parameters**
    * `doc` – DOM doc.

  Node

- `public` **Node**`( org.w3c.dom.Element` **elementNode** `)`
  - **Description**
    Node constructor with a DOM element as parameter.
  - **Parameters**
    * `elementNode` – A DOM element.

  Node

- `public` **Node**`( org.w3c.dom.Node` **node** `)`
  - **Description**
    Node constructor with a DOM node as parameter.
  - **Parameters**
    * `node` – A DOM node.

Methods

- `public java.util.List` **getAbsoluteChildren**`( ) throws MsqlException`
  - **Description**
    Returns the list of nodes that are children of the current context.
  - **Returns** – list the list of child nodes.
  - **Throws**
    * `at.ac.uni_linz.risc.mathbroker.msql.msqlSemantics.MsqlException` – If the given context is not vaild.

  getAttribute

- `public java.lang.String` **getAttribute**`( java.lang.String` **attributeName** `)`
  - **Description**
    Returns the attribute of the given name.
  - **Parameters**
    * `attributeName` – The name of the attribute.
  - **Returns** – string The coressponding attribute.

  getAttributeValue

- `public java.lang.String` **getAttributeValue**`( java.lang.String` **attributeName** `)`

- **Description**
  Returns the attribute value of the given attribute name.
- **Parameters**
  - `attributeName` – The name of an attribute.
- **Returns** – string The value of the attribute.

getChildByNumber

- `public java.util.List` **getChildByNumber**`( int` **number** `)`
  - **Description**
    Returns the node corresponding to the given number in the current context.
  - **Parameters**
    - `number` – The number of the node.
  - **Returns** – list The node is packed in a list.

getChildren

- `public java.util.List` **getChildren**`( java.lang.String` **name** `) throws MsqlException`
  - **Description**
    Returns the list of children of the current node having the given name as their names.
  - **Parameters**
    - `name` – The tag name of the required children.
  - **Returns** – list A list of child nodes of the current node having the given name as their names.

getChildrenByName

- `public java.util.List` **getChildrenByName**`( java.lang.String` **name** `)`
  - **Description**
    Returns the list of children of the given node.
  - **Parameters**
    - `name` – The tag name of the given node.
  - **Returns** – list A list of children of the given node.

getPosition

- `public int` **getPosition**`( )`
  - **Description**
    Returns an integer value corresponding to the position of the current node in the current context.
  - **Returns** – int An integer value corresponding to the position of the current node in the current context.

inChildren

- `public boolean` **inChildren**`( java.lang.String` **elementName** `)`
  - **Description**
    Checks if the element with the given name is a child of the current node.
  - **Parameters**
    - `elementName` – Name of the given element.

– **Returns** – true if the element is a child of the current node, false otherwise.

isElement

- `public boolean` **isElement**`( java.lang.Object` **object** `)`
  - **Description**
    Checks if the given object is an element in the current context.
  - **Parameters**
    * `object` – The given object.
  - **Returns** – true if the node is an element in the current context, false otherwise.

name

- `public java.lang.String` **name**`( )`

Class PathExprEvaluatorImpl

Evaluates path expressions.

Declaration public class PathExprEvaluatorImpl
extends java.lang.Object java.lang.Object
implements PathExprEvaluator

Constructors

- `public` **PathExprEvaluatorImpl**`( )`

Class PrimaryExprEvaluatorImpl

Implements methods to evaluate primary expressions. Primary expressions are var, literal, msqlFunction, dot, and parenthesized expression.

Declaration public class PrimaryExprEvaluatorImpl
extends java.lang.Object java.lang.Object
implements PrimaryExprEvaluator

Constructors

- `public` **PrimaryExprEvaluatorImpl**`( )`

Class RegularExprEvaluatorImpl

Declaration public class RegularExprEvaluatorImpl
extends java.lang.Object java.lang.Object
implements RegularExprEvaluator

Constructors

- `public` **RegularExprEvaluatorImpl**`( )`

Methods

- `public static Value` **evaluatePredicateExpr**`(` `at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST` **expr**`,` `Declaration` **declaration**`,` `Node` **node** `)` `throws java.io.IOException, JAXRException, JAXBException, MsqlException, java.lang.CloneNotSupportedException`
  - **Description**

    Implements the valuation function PE [[[E]]] d n to evaluate a predicate expression. It returns a set of MSDL document elements that satisfy the predicate.
  - **Parameters**
    - `expr` – The predicate expression.
    - `declaration` – The current declaration.
    - `node` – The current MSDL node.
  - **Returns** – value A set of nodes or empty set.
  - **Throws**
    - `at.ac.uni_linz.risc.mathbroker.msql.msqlSemantics.MsqlException` –
    - `java.lang.CloneNotSupportedException` –

Class SortComparator

Compares two Map.Entry objects for order based on the their keys.

Declaration public class SortComparator
extends java.lang.Object java.lang.Object
implements java.util.Comparator

Constructors

- `public` **SortComparator**`( )`

Methods

- `public int` **compare**`( java.lang.Object` **obj1**`, java.lang.Object` **obj2** `)`
  - **Description**

    Returns a negative integer, zero, or a positive integer if the first argument is less than, equal to, or greater than the second, result.

Class StringValue

Represents a string value.

Declaration public class StringValue
extends at.ac.uni_linz.risc.mathbroker.msql.msqlSemantics.Value Value

Constructors

- `public` **StringValue**`( java.lang.String` **value** `)`
  - **Parameters**
    - `value` –

METHODS

- `public java.lang.String` **getValue**`( )`
  - **Returns** – Returns the value.

  setValue
- `public void` **setValue**`( java.lang.String` **value** `)`
  - **Parameters**
    - `value` – The value to set.

CLASS VALUE

Super class for other value classes that other evaluate methods (syntactic functions) would return.

DECLARATION public abstract class Value
extends java.lang.Object java.lang.Object

ALL KNOWN SUBCLASSES BooleanValue

CONSTRUCTORS

- `public` **Value**`( )`

EXCEPTION MSQLEXCEPTION

MSQL exception handling.

DECLARATION public class MsqlException
extends java.lang.Exception java.lang.Exception

CONSTRUCTORS

- `public` **MsqlException**`( )` MsqlException
- `public` **MsqlException**`( java.lang.String` **message** `)`
  - **Parameters**
    - `message` –

  MsqlException
- `public` **MsqlException**`( java.lang.String` **message**`, java.lang.Throwable` **cause** `)`
  - **Parameters**
    - `message` –
    - `cause` –

  MsqlException
- `public` **MsqlException**`( java.lang.Throwable` **cause** `)`
  - **Parameters**
    - `cause` –

D.3 Package at.ac.uni_linz.risc.mathbroker.msql.treeWalker

CLASS ABSOLUTEPATHEXPR

This class handles (recognizes) absolute path expressions, i.e., path expressions which start with a step delimiter (/ or //) followed by a regular path expression.

DECLARATION public class AbsolutePathExpr
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

CONSTRUCTORS

- `public` **AbsolutePathExpr**`( )` AbsolutePathExpr
- `public` **AbsolutePathExpr**`( antlr.Token` **tok** `)`

METHODS

- `public ChildAST` **getChild**`( )`
  - **Returns** – The AST of the first child after the step delimiter.

  getFirst
- `public ChildAST` **getFirst**`( )`
  - **Returns** – AST of the first step expression, e.g., /step/next/etc.

  getNext
- `public ChildAST` **getNext**`( ChildAST` **child** `)`
  - **Returns** – AST tree of the next step expression, e.g., /step/next/etc.

  getPathSequence
- `public java.lang.String` **getPathSequence**`( )`
  - **Returns** – A string representation of the path expression after the first step delimiter and child expression. It is used for printing.

  getStepDel
- `public ChildAST` **getStepDel**`( )`
  - **Returns** – The AST of a step delimiter.

  print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

CLASS ALGORITHMNODE

This class handles (recognizes) Algorithm node.

DECLARATION public class AlgorithmNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

CONSTRUCTORS

- `public` **AlgorithmNode**`( antlr.Token` **tok** `)`

METHODS

- `public java.lang.String` **print**`( )`
  - **Description**
    returns the string representation of the token.

  toString
- `public java.lang.String` **toString**`( )`

Class AlgorithmStep

This class handels situations where an algorithm (which is part of the grammar) token occurs as a step in a path expression.

Declaration public class AlgorithmStep
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **AlgorithmStep**`( antlr.Token` **tok** `)`

Methods

- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class AndExpr

This class handels logical AND expressions.

Declaration public class AndExpr
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **AndExpr**`( antlr.Token` **tok** `)`

Methods

- `public ChildAST` **getLhs**`( )`
  - **Returns** – AST of the left hand side of the operator AND.

  getRhs
- `public ChildAST` **getRhs**`( )`
  - **Returns** – AST of the right hand side of the operator AND.

  print
- `public java.lang.String` **print**`( )`
  - **Description**
    Prints the AND expression.

  toString
- `public java.lang.String` **toString**`( )`

Class ArithmeticExpr

This class handels the AST of the Arithmetic node.

Declaration public class ArithmeticExpr
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **ArithmeticExpr**`( )` ArithmeticExpr
- `public` **ArithmeticExpr**`( antlr.Token` **tok** `)`

METHODS

- `public ChildAST` **getChild**`( )` getFirst
- `public ChildAST` **getFirst**`( )` getNext
- `public ChildAST` **getNext**`( ChildAST` **child** `)` getPredicateSequence
- `public java.lang.String` **getPredicateSequence**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

CLASS ASCENDINGNODE

This class handels the AST of the Ascending node.

DECLARATION public class AscendingNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

CONSTRUCTORS

- `public` **AscendingNode**`( antlr.Token` **tok** `)`

METHODS

- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

CLASS ASSIGNMENTNODE

This class handels the AST of the Assignment node. letAssignmentClause : var ASSIGNMENT∧ msqlExpr ;

DECLARATION public class AssignmentNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

CONSTRUCTORS

- `public` **AssignmentNode**`( antlr.Token` **tok** `)`

METHODS

- `public ChildAST` **getExprChild**`( )` getVarChild
- `public ChildAST` **getVarChild**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

CLASS ATNODE

This class handels the AST of the AT node.

DECLARATION public class AtNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

CONSTRUCTORS

- `public` **AtNode**`( antlr.Token` **tok** `)`

Methods

- `public ChildAST` **getAttribute**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class ChildAST

Represents AST structure of an MSQL query string. Every node in an MSQL AST tree has a left child, right child, and a sibling.

Declaration public abstract class ChildAST
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.MsqlAST MsqlAST

All known subclasses AbsolutePathExpr

Constructors

- `public` **ChildAST**`( )`

Methods

- `public MsqlAST` **leftChild**`( )` rightChild
- `public MsqlAST` **rightChild**`( )` sibling
- `public MsqlAST` **sibling**`( )`

Class ClassificationSlashNode

This class handels the AST node ClassificationSlash, i.e., when a slash occurs as part of a classification concept name.

Declaration public class ClassificationSlashNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **ClassificationSlashNode**`( antlr.Token` **tok** `)`

Methods

- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class ColonNode

This class handels the AST of the Colon node.

Declaration public class ColonNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **ColonNode**`( antlr.Token` **tok** `)`

Methods

- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class CommaNode

This class handels the AST of the Comma node.

Declaration public class CommaNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **CommaNode**`( antlr.Token` **tok** `)`

Methods

- `public ChildAST` **getSibling**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class ContainsNode

This class handels the AST of the Contains node.

Declaration public class ContainsNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **ContainsNode**`( antlr.Token` **tok** `)`

Methods

- `public ChildAST` **getExprChild**`( )` getFirst
- `public ChildAST` **getFirst**`( )` getNext
- `public ChildAST` **getNext**`( ChildAST` **child** `)` getSequence
- `public java.lang.String` **getSequence**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class CountNode

This class handels the AST of the Count node of the msqlFunction rule. msqlFunction : fname LPAR (exprList)? RPAR ; fname : COUNT — CONTAINS — EMPTY — POSITION — DOC — EXISTS; exprList : msqlExpr (COMMA msqlExpr)* ;

Declaration public class CountNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **CountNode**`( antlr.Token` **tok** `)`

Methods

- `public ChildAST` **getExprChild**`( )` getFirst
- `public ChildAST` **getFirst**`( )` getNext
- `public ChildAST` **getNext**`( ChildAST` **child** `)` getSequence
- `public java.lang.String` **getSequence**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class DescendingNode

This class handels the AST of the Descending node.

Declaration public class DescendingNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **DescendingNode**`( antlr.Token` **tok** `)`

Methods

- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class DocNode

This class handels the AST of the Doc node.

Declaration public class DocNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **DocNode**`( antlr.Token` **tok** `)`

Methods

- `public ChildAST` **getExprChild**`( )` getFirst
- `public ChildAST` **getFirst**`( )` getNext
- `public ChildAST` **getNext**`( ChildAST` **child** `)` getSequence
- `public java.lang.String` **getSequence**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class DotNode

This class handels the AST of the Dot node.

Declaration public class DotNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **DotNode**`( antlr.Token` **tok** `)`

Methods

- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class ElseNode

This class handels the AST of the Else node.

Declaration public class ElseNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **ElseNode**`( antlr.Token` **tok** `)`

Methods

- `public ChildAST` **getExprChild**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class EmptyNode

This class handels the AST of the Empty node.

Declaration public class EmptyNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **EmptyNode**`( antlr.Token` **tok** `)`

Methods

- `public ChildAST` **getExprChild**`( )` getFirst
- `public ChildAST` **getFirst**`( )` getNext
- `public ChildAST` **getNext**`( ChildAST` **child** `)` getSequence
- `public java.lang.String` **getSequence**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class EqualsNode

This class handels the AST of the Equals node.

Declaration public class EqualsNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **EqualsNode**`( antlr.Token` **tok** `)`

Methods

- `public ChildAST` **getLhs**`( )` getRhs
- `public ChildAST` **getRhs**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class EveryNode

This class handels the AST of the Every node.

Declaration public class EveryNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

CONSTRUCTORS

- `public` **EveryNode**`( antlr.Token` **tok** `)`

METHODS

- `public ChildAST` **getOtherSibling**`( )` getSibling
- `public ChildAST` **getSibling**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

CLASS EVERYQUANTIFIEDEXPR

This class handels the AST of the EveryQuantified node. quantifiedExpr : quantifier var IN msqlExpr (COMMA var IN msqlExpr)* SATISFIES msqlExpr ; quantifier : EVERY∧;

DECLARATION public class EveryQuantifiedExpr
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

CONSTRUCTORS

- `public` **EveryQuantifiedExpr**`( antlr.Token` **tok** `)`

METHODS

- `public ChildAST` **getFirst**`( )` getInChild
- `public ChildAST` **getInChild**`( )` getNext
- `public ChildAST` **getNext**`( ChildAST` **child** `)` getSatisfiesChild
- `public ChildAST` **getSatisfiesChild**`( )` getSequence
- `public java.lang.String` **getSequence**`( )` getVarChild
- `public ChildAST` **getVarChild**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

CLASS EXISTSNODE

This class handels the AST of the Exists node.

DECLARATION public class ExistsNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

CONSTRUCTORS

- `public` **ExistsNode**`( antlr.Token` **tok** `)`

METHODS

- `public ChildAST` **getExprChild**`( )` getFirst
- `public ChildAST` **getFirst**`( )` getNext
- `public ChildAST` **getNext**`( ChildAST` **child** `)` getSequence
- `public java.lang.String` **getSequence**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class FalseNode

This class handels the AST of the False node.

Declaration public class FalseNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **FalseNode**`( antlr.Token` **tok** `)`

Methods

- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class FromNode

This class handels the AST of the From node.

Declaration public class FromNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **FromNode**`( antlr.Token` **tok** `)`

Methods

- `public ChildAST` **getClassification**`( )` getClassificationSequence
- `public java.lang.String` **getClassificationSequence**`( )` getFirst
- `public ChildAST` **getFirst**`( )` getNext
- `public ChildAST` **getNext**`( ChildAST` **child** `)` getSlash
- `public ChildAST` **getSlash**`( )` getWhere
- `public ChildAST` **getWhere**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class GTENode

This class handels the AST of the GTE (Greater Than or Equal) node.

Declaration public class GTENode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **GTENode**`( antlr.Token` **tok** `)`

Methods

- `public ChildAST` **getLhs**`( )` getRhs
- `public ChildAST` **getRhs**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class GTNode

This class handels the AST of the GT (Greater Than) node.

Declaration public class GTNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **GTNode**`( antlr.Token` **tok** `)`

Methods

- `public ChildAST` **getLhs**`( )` getRhs
- `public ChildAST` **getRhs**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class IfExpr

This class handels the AST of the If node. ifExpr : IF∧ msqlExpr THEN msqlExpr ELSE msqlExpr

Declaration public class IfExpr
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **IfExpr**`( antlr.Token` **tok** `)`

Methods

- `public ChildAST` **getElseChild**`( )` getIfChild
- `public ChildAST` **getIfChild**`( )` getThenChild
- `public ChildAST` **getThenChild**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class ImplementationNode

This class handels the AST of the Implementation node.

Declaration public class ImplementationNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **ImplementationNode**`( antlr.Token` **tok** `)`

Methods

- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class ImplementationStep

This class handels situations where an implementation (which is part of the grammar) token occurs as a step in a path expression.

DECLARATION public class ImplementationStep
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

CONSTRUCTORS

- `public` **ImplementationStep**`( antlr.Token` **tok** `)`

METHODS

- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

CLASS INNODE

This class handels the AST of the In node.

DECLARATION public class InNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

CONSTRUCTORS

- `public` **InNode**`( antlr.Token` **tok** `)`

METHODS

- `public ChildAST` **getExprChild**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

CLASS LBRACKETNODE

This class handels the AST of the LBracket node.

DECLARATION public class LBracketNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

CONSTRUCTORS

- `public` **LBracketNode**`( antlr.Token` **tok** `)`

METHODS

- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

CLASS LETEXPR

This class handels the AST of the Let node.

DECLARATION public class LetExpr
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

CONSTRUCTORS

- `public` **LetExpr**`( antlr.Token` **tok** `)`

Methods

- `public ChildAST` **getAssignmentExpr**`( )` getFirst
- `public ChildAST` **getFirst**`( )` getInExpr
- `public ChildAST` **getInExpr**`( )` getNext
- `public ChildAST` **getNext**`( ChildAST` **child** `)` getSequence
- `public java.lang.String` **getSequence**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class LetterNode

This class handels the AST of the Letter node.

Declaration public class LetterNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **LetterNode**`( antlr.Token` **tok** `)`

Methods

- `public ChildAST` **getChild**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class LPARNode

This class handels the AST of the LPAR node.

Declaration public class LPARNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **LPARNode**`( antlr.Token` **tok** `)`

Methods

- `public ChildAST` **getChild**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class LTENode

This class handels the AST of the LTE node.

Declaration public class LTENode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **LTENode**`( antlr.Token` **tok** `)`

Methods

- `public ChildAST` **getLhs**`( )` getRhs
- `public ChildAST` **getRhs**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class LTNode

This class handels the AST of the LT node.

Declaration public class LTNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **LTNode**`( antlr.Token` **tok** `)`

Methods

- `public ChildAST` **getLhs**`( )` getRhs
- `public ChildAST` **getRhs**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class MachineNode

This class handels the AST of the Machine node.

Declaration public class MachineNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **MachineNode**`( antlr.Token` **tok** `)`

Methods

- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class MinusExpr

This class handels the AST of the Minus node.

Declaration public class MinusExpr
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **MinusExpr**`( antlr.Token` **tok** `)`

Methods

- `public ChildAST` **getLhs**`( )` getRhs
- `public ChildAST` **getRhs**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class MsqlAST

Represents top level of MSQL AST.

Declaration public abstract class MsqlAST
extends antlr.CommonAST antlr.CommonAST

All known subclasses AbsolutePathExpr

Constructors

- `public` **MsqlAST**`( )`

Methods

- `void` **initialize**`( antlr.collections.AST )` initialize
- `void` **initialize**`( int, java.lang.String )` initialize
- `void` **initialize**`( antlr.Token )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class MultExpr

This class handels the AST of the Mult node.

Declaration public class MultExpr
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **MultExpr**`( antlr.Token` **tok** `)`

Methods

- `public ChildAST` **getLhs**`( )` getRhs
- `public ChildAST` **getRhs**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class NENode

This class handels the AST of the NE (Not Equal) node.

Declaration public class NENode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **NENode**`( antlr.Token` **tok** `)`

Methods

- `public ChildAST` **getLhs**`( )` getRhs
- `public ChildAST` **getRhs**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class NotExpr

This class handels the AST of the Not node.

Declaration public class NotExpr
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **NotExpr**`( antlr.Token` **tok** `)`

Methods

- `public ChildAST` **getChild**`( )`
  - **Returns** – Returns the not.

  print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class NumberNode

This class handels the AST of the Number node.

Declaration public class NumberNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **NumberNode**`( antlr.Token` **tok** `)`

Methods

- `public ChildAST` **getChild**`( )` getSibling
- `public ChildAST` **getSibling**`( )` print
- `public abstract java.lang.String` **print**`( )` toInteger
- `public int` **toInteger**`( )` toString
- `public java.lang.String` **toString**`( )`

Class OrderbyNode

This class handels the AST of the Orderby node.

Declaration public class OrderbyNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **OrderbyNode**`( antlr.Token` **tok** `)`

Methods

- `public ChildAST` **getFirst**`( )` getMsqlExprChild
- `public ChildAST` **getMsqlExprChild**`( )` getNext
- `public ChildAST` **getNext**`( ChildAST` **child** `)` getSequence
- `public java.lang.String` **getSequence**`( )` getSortChild
- `public ChildAST` **getSortChild**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class OrExpr

This class handels the AST of the Or node.

Declaration public class OrExpr
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **OrExpr**`( antlr.Token` **tok** `)`

Methods

- `public ChildAST` **getLhs**`( )` getRhs
- `public ChildAST` **getRhs**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class PlusExpr

This class handels the AST of the plus node.

Declaration public class PlusExpr
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **PlusExpr**`( antlr.Token` **tok** `)`

Methods

- `public ChildAST` **getLhs**`( )` getRhs
- `public ChildAST` **getRhs**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class PositionNode

This class handels the AST of the Position node.

Declaration public class PositionNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **PositionNode**`( antlr.Token` **tok** `)`

Methods

- `public ChildAST` **getExprChild**`( )` getFirst
- `public ChildAST` **getFirst**`( )` getNext
- `public ChildAST` **getNext**`( ChildAST` **child** `)` getSequence
- `public java.lang.String` **getSequence**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class PrimaryExpr

This class handels the AST of the Primary node.

Declaration public class PrimaryExpr
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **PrimaryExpr**`( )` PrimaryExpr
- `public` **PrimaryExpr**`( antlr.Token` **tok** `)`

Methods

- `public ChildAST` **getExpr**`( )` getFirst
- `public ChildAST` **getFirst**`( )` getNext
- `public ChildAST` **getNext**`( ChildAST` **child** `)` getPredicateSequence
- `public java.lang.String` **getPredicateSequence**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class ProblemNode

This class handels the AST of the Problem node.

Declaration public class ProblemNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **ProblemNode**`( antlr.Token` **tok** `)`

Methods

- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class ProblemStep

This class handels the AST of the ProblemStep node. It used for situations where problem occurs as a step in a path expression.

Declaration public class ProblemStep
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **ProblemStep**`( )` ProblemStep
- `public` **ProblemStep**`( antlr.Token` **tok** `)`

Methods

- `public java.lang.String` **getS**`( )`
  - **Returns** – Returns the s.

  print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class RBracketNode

This class handels the AST of the RBracket node.

Declaration public class RBracketNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **RBracketNode**`( antlr.Token` **tok** `)`

Methods

- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class RegularExpr

This class handels the AST of the RegularExpr node.

Declaration public class RegularExpr
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **RegularExpr**`( )` RegularExpr
- `public` **RegularExpr**`( antlr.Token` **tok** `)`

Methods

- `public ChildAST` **getExprChild**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class RelativePathExpr

This class handels the AST of the RelativePathExpr node.

Declaration public class RelativePathExpr
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **RelativePathExpr**`( )` RelativePathExpr
- `public` **RelativePathExpr**`( antlr.Token` **tok** `)`

Methods

- `public ChildAST` **getChild**`( )` getFirst
- `public ChildAST` **getFirst**`( )` getNext
- `public ChildAST` **getNext**`( ChildAST` **child** `)` getPathSequence
- `public java.lang.String` **getPathSequence**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class ReturnNode

This class handels the AST of the Return node.

Declaration public class ReturnNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **ReturnNode**`( antlr.Token` **tok** `)`

Methods

- `public ChildAST` **getEntity**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class RPARNode

This class handels the AST of the RPAR node.

Declaration public class RPARNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **RPARNode**`( antlr.Token` **tok** `)`

Methods

- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class SatisfiesNode

This class handels the AST of the Satisfies node.

Declaration public class SatisfiesNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **SatisfiesNode**`( antlr.Token` **tok** `)`

Methods

- `public ChildAST` **getChild**`( )` getExprChild
- `public ChildAST` **getExprChild**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class SelectNode

This class handels the AST of the Select node.

Declaration public class SelectNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **SelectNode**`( antlr.Token` **tok** `)`

Methods

- `public java.lang.String` **getClassification**`( )` getEntity
- `public ChildAST` **getEntity**`( )` getFromOrWhere
- `public java.lang.String` **getFromOrWhere**`( )` getOrderbyChild
- `public ChildAST` **getOrderbyChild**`( )` getOrderbyChildText
- `public java.lang.String` **getOrderbyChildText**`( )` getOrderbyExpr
- `public ChildAST` **getOrderbyExpr**`( )` getSomeOrEvery
- `public ChildAST` **getSomeOrEvery**`( )` getSort
- `public java.lang.String` **getSort**`( )` getWhereChild
- `public ChildAST` **getWhereChild**`( )` getWhereChildText
- `public java.lang.String` **getWhereChildText**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class ServiceNode

This class handels the AST of the Service node.

Declaration public class ServiceNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **ServiceNode**`( antlr.Token` **tok** `)`

Methods

- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class ServiceStep

This class handels the AST of the ServiceStep node.

DECLARATION public class ServiceStep
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

CONSTRUCTORS

- `public` **ServiceStep**`( antlr.Token` **tok** `)`

METHODS

- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

CLASS SLASHNODE

This class handels the AST of the Slash node.

DECLARATION public class SlashNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

CONSTRUCTORS

- `public` **SlashNode**`( antlr.Token` **tok** `)`

METHODS

- `public ChildAST` **getStep**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

CLASS SLASHSLASHNODE

This class handels the AST of the SlashSlash node.

DECLARATION public class SlashSlashNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

CONSTRUCTORS

- `public` **SlashSlashNode**`( antlr.Token` **tok** `)`

METHODS

- `public ChildAST` **getSibling**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

CLASS SOMENODE

This class handels the AST of the Some node.

DECLARATION public class SomeNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

CONSTRUCTORS

- `public` **SomeNode**`( antlr.Token` **tok** `)`

Methods

- `public ChildAST` **getOtherSibling**`( )` getSibling
- `public ChildAST` **getSibling**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class SomeQuantifiedExpr

msqlExpr (COMMA var IN msqlExpr)* SATISFIES msqlExpr ; quantifier : SOME∧ ;

Declaration public class SomeQuantifiedExpr
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **SomeQuantifiedExpr**`( antlr.Token` **tok** `)`

Methods

- `public ChildAST` **getFirst**`( )` getInChild
- `public ChildAST` **getInChild**`( )` getNext
- `public ChildAST` **getNext**`( ChildAST` **child** `)` getSatisfiesChild
- `public ChildAST` **getSatisfiesChild**`( )` getSequence
- `public java.lang.String` **getSequence**`( )` getVarChild
- `public ChildAST` **getVarChild**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class Step

This class handels the AST of the Step node.

Declaration public class Step
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **Step**`( )` Step
- `public` **Step**`( antlr.Token` **tok** `)`

Methods

- `public ChildAST` **getChild**`( )` getFirstPredicate
- `public ChildAST` **getFirstPredicate**`( )` getLBracket
- `public ChildAST` **getLBracket**`( )` getNext
- `public ChildAST` **getNext**`( ChildAST` **child** `)` getPredicateSequence
- `public java.lang.String` **getPredicateSequence**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class StringNode

This class handels the AST of the In node.

Declaration public class StringNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **StringNode**`( antlr.Token` **tok** `)`

Methods

- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class ThenNode

This class handels the AST of the Then node.

Declaration public class ThenNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **ThenNode**`( antlr.Token` **tok** `)`

Methods

- `public ChildAST` **getExprChild**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class TrueNode

This class handels the AST of the True node.

Declaration public class TrueNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **TrueNode**`( antlr.Token` **tok** `)`

Methods

- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

Class VarExpr

This class handels the AST of the In node. var : DOLLAR∧ qName;

Declaration public class VarExpr
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

Constructors

- `public` **VarExpr**`( antlr.Token` **tok** `)`

METHODS

- `public ChildAST` **getVarName**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

CLASS WALKER

This class is a demo class for MSQL grammar parsing and walking. So it accepts a query string, parses it, and if correct prints the parsed grammar out.

DECLARATION public class Walker
extends java.lang.Object java.lang.Object

CONSTRUCTORS

- `public` **Walker**`( )`

METHODS

- `public static void` **main**`( java.lang.String[]` **args** `) throws java.lang.Exception`

CLASS WHERENODE

This class handels the AST of the Where node.

DECLARATION public class WhereNode
extends at.ac.uni_linz.risc.mathbroker.msql.treeWalker.ChildAST ChildAST

CONSTRUCTORS

- `public` **WhereNode**`( antlr.Token` **tok** `)`

METHODS

- `public ChildAST` **getMsqlExpr**`( )` getReturn
- `public ChildAST` **getReturn**`( )` print
- `public abstract java.lang.String` **print**`( )` toString
- `public java.lang.String` **toString**`( )`

References

[1] FIPS PUB 127-2, *Structured Query Language*, FIPS, June 1993, See http://www.itl.nist.gov/fipspubs/fip127-2.htm.

[2] Rebhi Baraka, *A Foundation for a Mathematical Web Services Query Language: A Survey on Relevant Query Languages and Tools*, Tech. report, RISC-Linz, Austria, September 2004, See http://apache.risc.uni-linz.ac.at/internals/ActivityDB/publications/download/risc_2036/04-18.ps.gz.

[3] Rebhi Baraka, Olga Caprotti, and Wolfgang Schreiner, *A Registry Service as a Foundation for Brokering Mathematical Services*, Tech. report, RISC-Linz, Austria, February 2004, See ftp://ftp.risc.uni-linz.ac.at/pub/techreports/2004/04-13.ps.gz.

[4] _____, *A Web Registry for Publishing and Discovering Mathematical Services*, Proceedings of IEEE Conference on e-Technology, e-Commerce, and e-Service (Hong Kong Baptist University, Hong Kong, March 29 – April 1), IEEE Computer Society, Los Alamitos, CA, 2005.

[5] Olga Caprotti and Wolfgang Schreiner, *Towards a Mathematical Service Description Language*, International Congress of Mathematical Software ICMS 2002 (Bejing, China, August 17–19), World Scientific Publishing, Singapore, 2002.

[6] Don Chamberlin, Jonathan Robie, and Daniela Florescu, *QUILT: An XML Query Language for Heterogeneous Data Sources*, Proceedings of WebDB 2000 Conference, in Lecture Notes in Computer Science.

[7] Byron Choi, Mary Fernández, and Jérôme Siméon, *The XQuery Formal Semantics: A Foundation for Implementation and Optimization*, Tech. report, AT&T Labs – Research, May 2002.

[8] World Wide Web Consortium, *Document Object Model (DOM)*, W3C Recommendation, September 2004, See http://www.w3.org/DOM/.

[9] _____, *OWL Web Ontology Language Reference*, W3C Recommendation, February 2004, See http://www.w3.org/TR/2004/REC-owl-ref-20040210/.

[10] _____, *Resource Description Framework (RDF)*, W3C Recommendation, September 2004, See http://www.w3.org/RDF/.

[11] _____, *XML Path Language (XPath) 2.0*, W3C Working Draft, April 2005, See http://www.w3.org/TR/xpath20/.

[12] _____, *XQuery 1.0: An XML Query Language*, W3C Working Draft, April 2005, See http://www.w3.org/TR/xquery/.

[13] _____, *XQuery 1.0 and XPath 2.0 Data Model*, W3C Working Draft, April 2005, See http://www.w3.org/TR/xpath-datamodel/.

[14] _____, *XQuery 1.0 and XPath 2.0 Formal Semantics*, W3C Working Draft, June 2005, See http://www.w3.org/TR/xquery-semantics/.

[15] I. Dahn and A. Asperti, *Mathematical Knowledge Management and Searchability*, Deliverable D5.4, 2001, See http://monet.nag.co.uk/mkm/MKMNetTN-D5-4.pdf.

[16] *ebXML Registry Reference Implementation Project (ebxmlrr)*, April 2004, See http://ebxmlrr.sourceforge.net.

[17] *ebXML Registry Information Model v2.0*, OASIS, December 2001, See http://www.oasis-open.org/committees/regrep/documents/2.0/specs/ebRIM.pdf.

[18] *ebXML Registry Services Specification v2.0*, OASIS, April 2002, See http://www.oasis-open.org/committees/regrep/documents/2.0/specs/ebRS.pdf.

[19] Peter Fankhauser, *Xquery formal semantics state and challenges*, SIGMOD Record **30** (2001), no. 3, 14 – 19.

[20] *Galax* , March 2005, See http://www.galaxquery.org/.

[21] Jan Hidders, Jan Paredaens, Roel Vercammen, and Serge Demeyer, *A Light but Formal Introduction to XQuery*, Proceedings of the Second International XML Database Symposium (XSym 2004) (Toronto, Canada, August 29 – 30), Springer-Verlag, 2004.

[22] *IPSI-XQ – The XQuery Demonstrator*, September 2005, See http://www.ipsi.fraunhofer.de/oasys/projects/ipsi-xq/index_e.html.

[23] *MathBroker — A Framework for Brokering Distributed Mathematical Services*, Reseach Institute for Symbolic Computation (RISC), April 2004, See http://www.risc.uni-linz.ac.at/projects/basic/mathbroker.

[24] *MONET — Mathematics on the Web*, The MONET Consortium, April 2004, See http://monet.nag.co.uk.

[25] *Mathematical Services Description Language (MSDL)*, Reseach Institute for Symbolic Computation (RISC), April 2004, See http://poseidon.risc.uni-linz.ac.at:8080/mathbroker/results/xsd.html.

[26] William Naylor and Julian Padget, *Semantic Matching for Mathematical Services*, Proceedings of the Forth International conference on Mathematical Knowledge Management (Bremen, Germany, 15 – 17 July), Springer, 2005.

[27] Terence Parr, *ANTLR (ANother Tool for Language Recognition) Reference Manual*, September 2005.

[28] *MathBroker Registry API*, Reseach Institute for Symbolic Computation (RISC), April 2004, See http://poseidon.risc.uni-linz.ac.at:8080/results/registry/MBregistryAPI/.

[29] David A. Schmidt, *Denotational Semantics – A Methodology for Language Development*, Allyn and Bacon, Boston, 1986.

[30] J. E. Stoy, *Denotational Semantics: The Scott-Strachey Aproach to Programming Language Theory*, Cambridge, Massachusetts, 1977.

[31] Philip Wadler, *A Formal Semantics of Patterns in XSL*, submitted to Markup Technologies 99, January 1999, See http://www.cs.bell-labs.com/ wadler/topics/xml#xsl-semantics.

[32] ———, *Two Semantics for XPath*, Note, January 2000, See http://homepages.inf.ed.ac.uk/wadler/papers/xpath-semantics/xpath-semantics.pdf.

[33] *XQEngine* , March 2005, See http://xqengine.sourceforge.net/.

[34] *XQuery API for Java (XQJ)*, Java Community Process, July 2004, See http://jcp.org/aboutJava/communityprocess/edr/jsr225/index.html.