SISU dokument

CQL

– a query language for retrieving multimedia information

Peter Rosengren

nr
12

1. Introduction

Today corporate users need to access and combine information from a number of information sources. Some business information is stored in relational databases, while other important information is accessed through searches in a text retrieval system. Additional sources of information may be picture and drawings archives. Each information source might have its own data model and access mechanism, or query language, causing severe usability problems.

The objective of the Intuitive project is to provide efficient and easy to use tools for end users to access information in heterogeneous corporate databases.

Within the project we are investigating users and application requirements to define a generic architecture for Information Retrieval from heterogeneous databases.

The project also address the needs of application designers in developing methods and tools for customising the generic software architecture for different applications. In this sense Intuitive can be seen as both a generic retrieval system as well as a system for creating information retrieval applications.

Rosengren et al have already given an overview of this work [Rosengren93a], [Rosengren93b], [Rosengren93c]. Wingstedt et al discuss the requirements on the functionality of end-user tools for Information Retrieval and their user interface [Wingstedt93]. Bern et al address the needs of application developers in describing a first prototype implementation of the Tools applied in three different applications [Bern93]. The reader that is unfamiliar with the Intuitive Project is referred to these reports.

The purpose of this document is to describe a first version of the internal query language used within Intuitive. This language is called Conceptual Query Language, CQL.

The purpose of CQL is to allow the Intuitive Tools and other modules such as the Dialogue Manager to represent users' queries across a heterogeneous information space. A CQL query is expressed in terms of restrictions on entities, attributes, relationships and value contents that are known by the Intuitive System.

Within Intuitive we employ a 3-schema level framework [Rosengren93a], [Rosengren93c]:

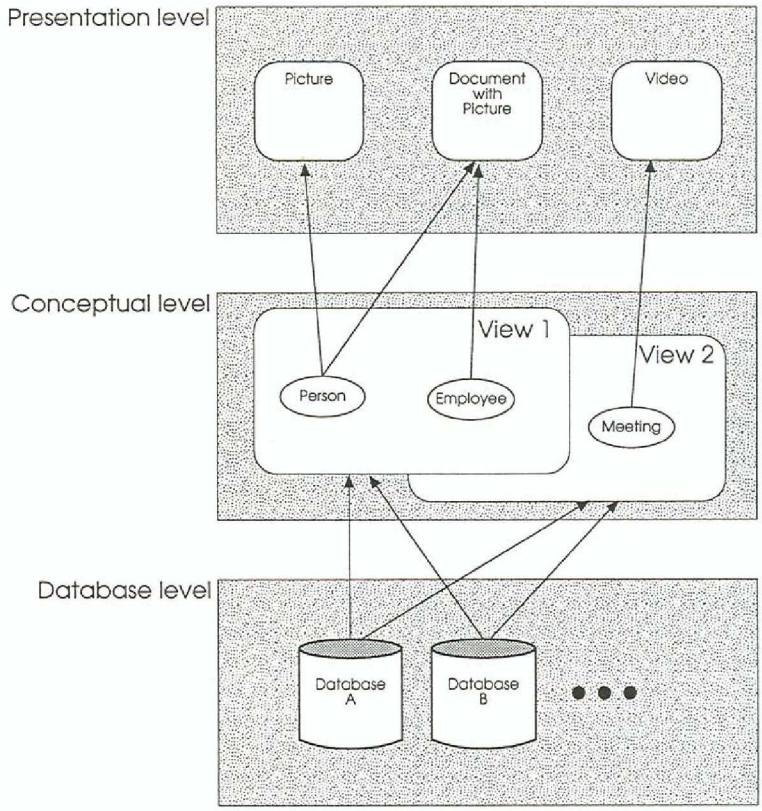

Figure 1.1 A 3-schema level framework.

At the *database level* each database has a logical database model and a specific access language - examples are a relational database with SQL, a document category system with free text search, a picture database with keyword searches.

The *conceptual level* constitutes the conceptual modelling of real world objects within a particular application. Information about an object at this level might be distributed over several databases, for instance information about a Project Entity could be project data stored in a relational databases, project description found in a full text database and a picture with project participants in a picture database.

At the *presentation level* presentation aspects are described.

The scope of CQL is the conceptual level, i.e. CQL queries are expressed in terms of concepts defined at this level. This implies that applications issuing CQL queries can access data from various databases without concern of their specific characteristics, e.g. regarding structure and access language.

The document is organised as follows. Chapter 2 discusses the requirements on CQL. Chapter 3 describes the language. Finally, in Chapter 4 we give examples of how the language is used.

2. Requirements on CQL

Before describing CQL, we need to consider the requirements on the language. Two questions immediately arise:

- Who will produce a query?
- What do we need to express in a query?

Within Intuitive queries will be produced in three different ways:

- Directly by a user through the *visual query system.*
- Directly by a user through the *natural language interface.*
- By the *Dialogue Manager* that constructs a CQL-query based on what the user does and says.

When queries are formulated through the visual query interface, the system can restrict the user queries by displaying the structure of the conceptual model and only accept queries that are based on entity classes and relationships known by the system. This type of query we call a *conceptual query* since it asks about information stating restrictions on known concepts, i.e. entity classes, relationship classes and attribute classes.

When queries are formulated through the natural language interface there of course will be precise queries but we also must expect queries that are fuzzy and imprecise. Users might ask queries like "are there any projects about multimedia databases?", without specifically stating which entity classes are involved in the query or which attributes to be restricted.

Fuzzy and imprecise queries are common in text retrieval system but rarely supported in traditional database environments. Conventional database languages assume that the system has complete knowledge about the data in the database, i.e. the database schema completely describes every instance in the database [Fuhr93]. To be able to answer the example query above a conventional DBMS would require that the entity Project has an attribute that explicitly describes its type, category or subject, but there would still be no way to rank entities according to how relevant they are with respect to the user's query. In CQL we introduce a special function called "about" to handle imprecise queries. The about-function can be seen as a *vague predicate* [Fuhr90].

In some cases a user or a module might be able to specify an attribute name and a formal restriction but not the corresponding entity. An example is a user that knows that "Intuitive" is a name but not what it is a name of, e.g. a company, a project or a document. This would require a query that can be evaluated over all entity classes that have an attribute called "name".

In a more extreme case a user might only know a value but not what it stands for and wants to find out what the system knows about this value, i.e. the system needs to support *access by value* [Motro86].

Motro developed the system Baroque which offered the user four functions to scan a database which were called "What is it?", "What is known about it?", "What is the connection?" and "Any others like it?".

In Baroque it was possible to ask questions like:

```
> "What is Mozart?"
System answer: Mozart is NAME of COMPOSER, AUTHOR of COMPOSITION

> "What is known about Mozart?"
System answer: Mozart is
NAME of COMPOSER having
      COUNTRY Austria
      PERIOD Classical
      YEAR_OF_BIRTH 1756
      YEAR_OF_DEATH 1791
AUTHOR of COMPOSITION having IDENTIFICATION
      (Hunt, Mozart)
      (Jupiter, Mozart)
      (Magic_Flute, Mozart)
```

Queries like this were supported by an extra table in the database storing pairs of values and attributes. To support features like this we need to introduce the concept of unbound variables in CQL that can be bound to different entity classes and attribute classes at evaluation time. Given unbound variables in CQL and the Dictionary Structure of Intuitive [Rosengren93c] access by value is easy to implement within Intuitive.

We will refer to this type of queries as *variable queries*.

Finally in a multimedia environment it will be natural to ask queries about the content of the multimedia objects rather than their conceptual classification. For a text document this could mean proximity searching, phrase searching etc. For images this could mean pattern matching of a bitmap against the image. This type of query is called a *content query*.

Below we will list the requirements for the various types of querying that we are supporting. A general requirement is of course that it should be possible to mix all modes of searching within one query.

2.1. Conceptual querying:

It must be possible to ask for entities from a certain entity class by giving logical constraints of arbitrary complexity on its attributes.

It must be possible to express constraints about the relationships between entities. For instance it should be possible to ask for entities having three relationships of a certain Relationship class to entities for which other constraints are supplied. In this way chains of combined entity restrictions and relationship constraints can be formed.

2.2. Vague querying

It should be possible to ask if an entity is associated with a set of linguistic terms, like "multimedia, databases, GUI, MMDBMS" [Rosengren93d]. This corresponds to relevance searching found in most modern text retrieval system and could also be applied to keyword based systems containing pictures, videos, drawings, slides etc.

2.3. Variable querying

It should be possible to express a query that evaluates over several entity classes.

It should be possible to express a constraint that evaluates over several attributes of a specific entity class.

It should be possible to express a query that evaluates over several relationships for a particular entity class.

2.4. Content querying:

For multimedia objects it must be possible to express constraints about their content. Advanced text searching should be supported including *proximity search*, *phrase searching* and use of *thesaurus*.

It should also be possible to use *externally* defined functions that are evaluated over the objects in a database.

3. Language Description

CQL uses, like most query languages, a basic "select-from-where" syntax. Other query languages for querying about multimedia objects have used other syntactical structures. In the Multos Query Language the following syntax where used for querying about multimedia documents in an office filing system [Bertino90].

```
FIND DOCUMENTS [VERSION version-clause]
      [SCOPE scope-clause]
      [TYPE type-clause]
      WHERE condition-clause
```

Celanto et al also report about a system for querying multimedia documents in an office environment [Celanto91]. Their query syntax is:

retrieve class_name (**types**|**instances**)

[**in** *partition*]

[**with** *conceptual_description*]

[**environment** *environment_description*]

[**contents** *contents_description*]

We think it is unnecessary to depart from the conventional select-from-where-syntax. The basic CQL-statement therefore look like a SQL-query:

```
SELECT *
FROM Person
WHERE Person.Age > 20 AND
Person.FirstName = "Peter";
```

The CQL-syntax differs from the SQL-syntax in that CQL makes entity constraints and relationship constraints explicit. A query that involves constraints on two entity classes and a relationship between the entity classes looks like:

```
SELECT Person.Name
FROM Person
WHERE Person.Age > 20

SELECT *
FROM Project
WHERE Project.StartDate >= 920518

RELATIONSHIPS
Person WorksIn Project
;
```

In SQL this would be expressed in one compound statement. It would of course also be possible to define CQL in that way but we think this syntax favours clarity. A full description of the syntax is given in the appendix.

Moreover, the module that produces the CQL query does not have to have knowledge about how a relationship is calculated, it merely have to state that the relationship should hold.

Also, this syntactic form is more suitable for the Query Splitter that will take a CQL query and split it into subqueries which are sent to various databases. Note that the performance of the system will not be determined by the order of the two sub-queries in CQL. Performance will rely on the Query Splitter and its internal modules.

In this section we will explain the various constructs in CQL and their semantics.

3.1. Query Specification

A query consists of two parts:

- A set of entity set specifications each specifying a subset of instances to be retrieved from one entity class and which attributes to show.
- A set of relationships that should hold between entity instances specified in the entity set specifications part.

A query can be seen as a function that takes a conceptual schema as input and maps it into a subset of that schema.

3.2. Entity Set Specification

An Entity Set Specification defines a set of entities within the databases that fulfil some restriction.

The specification of an entity set to be retrieved consists of four parts:

- A specification of the output to be retrieved from the entity.
- Name of the entity class to be searched.
- Restrictions on attributes for each entity in that class.
- Restrictions specifying "about" criteria.

```
SELECT *
FROM Person
WHERE Person.Name = "Peter"
```

This retrieves all entities belonging to entity class Person where attribute "Name" equals "Peter".

If no restrictions for attributes are given all entities of the class are returned.

Output can be specified by giving a list of the attributes to be retrieved. The "*" character has the usual meaning of "select all". An empty output specification is also possible.

It is possible to use unbound variables in an Entity Set Specification:

```
SELECT *
FROM X
WHERE X.Name = "Peter"
```

The above Entity Set Specification is instantiated into one Entity Set Specification for each entity class that has an attribute called "Name". It is possible to control the instantiation by giving a list of entity classes to be considered, see example below:

```
SELECT *
FROM X [Project, Result]
WHERE X.Name = "Intuitive"
```

Variables can also be used at the attribute level. The use of variables makes it easy to implement *access by value* [Motro86] within Intuitive.

```
SELECT *
FROM X
WHERE X.Y = "Peter Rosengren"
```

An Entity Set Specification is a function that takes an entity class as input and maps it into a subclass of that entity class.

3.3. Attribute Restrictions

Compound attribute restrictions are allowed with the normal nesting of logical connectives:

```
SELECT *
FROM Person
WHERE (Person.name = "Peter" AND Person.age = 25) OR ( Person.name =
"Carl" AND Person.age > 20)
```

The above query will retrieve all entities from Entity class Person that either has a name equal to Peter and an age equal to 25, or a name equal to Carl and an age that is over 20.

An attribute can be compared with an another attribute belonging to the same entity or to another related entity.

The restrictions that are possible to express for an attribute will depend on its data type.

3.4. Vague Restrictions

```
SELECT *
FROM Report
WHERE Report.About(0.75, "user interface", "design")
```

The about function accepts a *relevance threshold value* and a set of terms as input. It calculates a relevance measure between 0 and 1 for each entity and assigns this measure to the entity. It then orders the entity set with respect to this relevance measure. It then truncates the ordered entity list with respect to the given threshold value. Note that it is

possible to leave out the threshold value and in that way get a relevance measure for all entities in the class.

About can be seen as a *vague predicate* [Fuhr90]. Other vague predicates are possible such as "at most", "high", "low" and "some". These are not included in the first version of CQL but the language can be extended to handle other vague predicates.

Normal attribute restrictions can be mixed with vague predicates. If this is done, the attribute restrictions have the highest precedence, i.e. only entities fulfilling the attribute criteria are considered for evaluation of the about restriction.

Consider the following query:

```
SELECT *
FROM Project
WHERE Project.StartDate > 920101
AND
Project.About(0.75, "user interface", "design")
```

This query will first find all projects that have started after 1 January 1992 and then for each how them calculate the to which extent they have to do with user interface design. This means that a project with very high relevance for user interface design but with a start date in 1991 will not be included in the answer.

Another way to put this is that attributes represents known facts about an entity and that facts always should be considered first before dealing with vague ideas.

3.5. Relationship Restrictions

Three types of constraints can be expressed about relationships:

- That they exist.
- That they do not exist.
- That they have a certain cardinality.

A relationship restriction is always evaluated within the context of two entity set specifications:

```
SELECT *
FROM Person
WHERE name="Peter" AND age > 20

SELECT *
FROM Car
WHERE year = 1980

RELATIONSHIPS
Person owns(3) Car
```

The above statement will retrieve all entities from entity class Person which has three "owns-relations" to entities within the entity set "Cars made in year 1980". Note that if a person owns three cars but only two made 1980 it will not be part of the answer.

Note also if a person owns four or more cars made 1980 it will not be part of the answer.

The expression owns() means that there exists some relationship. It could have been expressed by owns(>=1).

4. Query Examples

In this section we will give some examples of how CQL works. We will go through examples to show how CQL meets the requirements for each of the search modes discussed above, i.e. *conceptual querying*, *vague querying*, *variable querying* and *content querying*.

Each query example will first be expressed as a natural language phrase and then a corresponding CQL statement. Note that this is only for ease of reading, it does not specify the rules for translation of natural language statements into CQL.

4.1. Conceptual Querying

Assume the following simple conceptual model:

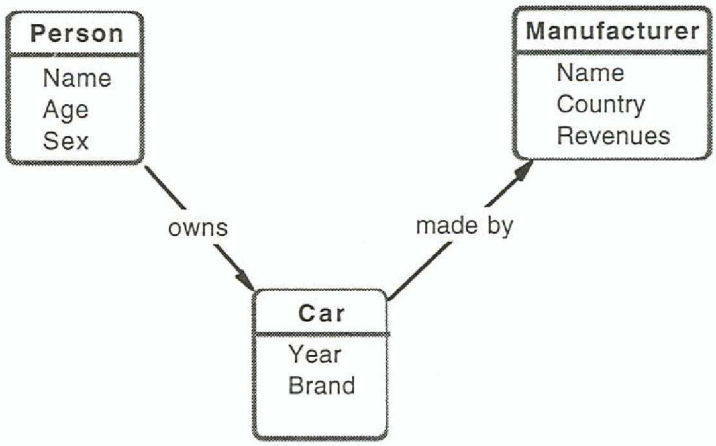

Figure 4.1 Example conceptual model

Below we will give some examples of user queries and how they are expressed in CQL.

Query: Give me all persons named Peter that are older than 20 years.

```
SELECT *
FROM Person
WHERE Person.name="Peter" AND Person.age > 20
```

Query: Give me the age of all persons named Peter that are older than 20 years and that owns a car made after 1985.

```
SELECT Person.Age
FROM Person
WHERE Person.name="Peter" AND Person.age > 20

SELECT
```

```
FROM Car
WHERE Car.year > 1985

RELATIONSHIPS
Person owns() Car
```

Query: Give me all persons named Peter that are older than 20 years and that owns a car made before 1980 by a Japanese manufacturer. Also, display the name of the Manufacturer.

```
SELECT *
FROM Person
WHERE Person.name="Peter" AND Person.age > 20

SELECT
FROM Car
WHERE Car.year < 1980

SELECT Manufacturer.Name
FROM Manufacturer
WHERE Manufacturer.country = "Japan"

RELATIONSHIPS
Person owns() Car
Car madeBy() Manufacturer
```

4.2. Vague Querying

Assume the following conceptual model:

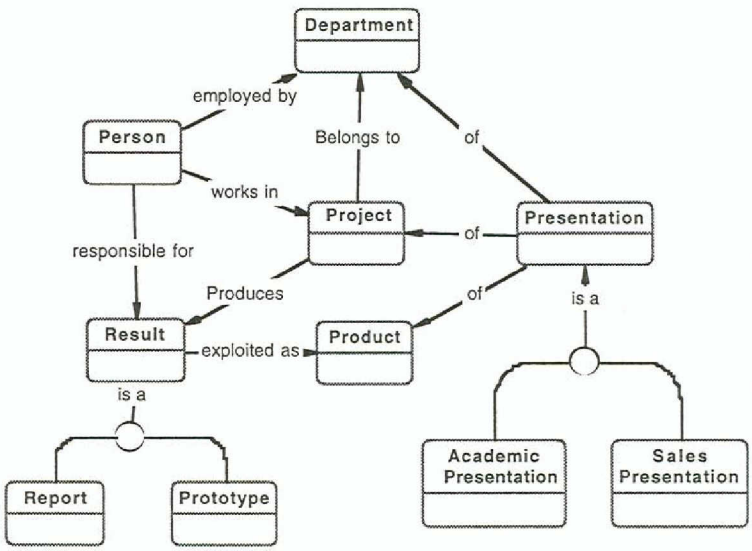

Figure 4.2 Example conceptual model

Also assume that Person, Project, Department and Product are stored in a relational database, while Academic Presentation and Report are to be found in a text database and Sales Presentation as well as Prototype are stored in a video database. Report and Prototype are subclasses of Result while Academic Presentation and Sales Presentation are subclasses of Presentation. The attributes of the entity classes are not shown in the figure above.

Note that the first parameter to the about-function is the required threshold value. This could be any number provided by the system or the user. We will use 0.75 (75 percent relevance) as a default value throughout the examples.

Query: Give me all results about visual languages for multimedia databases

```
SELECT *
FROM Result
WHERE Result.About(0.75, "multimedia", "databases", "visual
language")
```

Query: Give me all information concerning projects, that have started after 920101 and have produced reports about multimedia databases

```
SELECT *
FROM Project
WHERE Project.StartDate > 920101

SELECT
FROM Report
WHERE Report.About(0.75, "multimedia", "databases")

RELATIONSHIPS
Project Produces() Report
```

Query: Give me all presentation material about the Dialogue Manager in the Intuitive project

```
SELECT *
FROM Presentation
WHERE Presentation.About(0.75, "dialogue", "intelligent",
"cooperative", "dialogue manager")

SELECT
FROM Project
WHERE Project.Name = "Intuitive"

RELATIONSHIPS
Presentation of() Project
```

4.3. Variable Querying

We will assume the same conceptual model as in the previous section.

Query: Do you have any presentations or reports about user interface design?

```
SELECT *
FROM X [Presentation, Report]
WHERE X.About(0.75, "user interface", "design")
```

Query: What do you know about graphical user interfaces and query language design?

```
SELECT *
FROM X
WHERE X.About(0.75, "user interface", "graphical")
OR X.About(0.75, "query language", "design")
```

Query: Show me all projects that do not have anything to do with multimedia but are related to something that has to do with multimedia databases.

```
SELECT *
FROM Project
WHERE NOT Project.About(0.75, "multimedia")

SELECT
FROM X
WHERE X.About(0.75, "multimedia", "databases")

RELATIONSHIPS
Project Y X
```

Query: Is there any Product that corresponds to the abbreviation XPMG?

```
SELECT *
FROM Product
WHERE Product.X = "XPMG"
```

Query: What do you have that is named Roland?

```
SELECT *
FROM X
WHERE X.Name = "Roland"
```

Query: What is Roland?

```
SELECT *
FROM X
WHERE X.Y = "Roland"
```

Query: Is there anything at all about Roland?

```
SELECT *
```

```
FROM X
WHERE X.Y = "Roland"
OR X.About(0.75, "Roland")
```

4.4. Content Querying

In this section we will give some examples of how content expressions are formulated.

4.4.1. Text

Proximity

Query: Retrieve all project descriptions that contain the word databases within a distance of 5 words from multimedia.

```
SELECT *
FROM Project
WHERE Project.Description CONTAINS TEXT "databases" WITHINW5
"multimedia"
```

Free Text

Query: Retrieve all project descriptions that contain the phrase "multimedia databases need graphical interfaces".

```
SELECT *
FROM Project
WHERE Project.Description CONTAINS TEXT "multimedia databases need
graphical interfaces"
```

Any Order

Query: Retrieve all project descriptions that contain the two words "multimedia" and "database" in any order. Allow all different inflections of "database".

```
SELECT *
FROM Project
WHERE Project.Description CONTAINS TEXT ANY STEM "database",
"multimedia"
```

4.4.2. Other Media Types

For other media types than text, there currently are three operators available. CONTAINS checks if the media object is contained within another object. PART_OF checks if the media object is part of another object. LIKE checks if the media object is similar to another object. In all these cases we rely on the underlying database to provide such a comparison facility.

Assume that the attribute "description" is an image. Then, the following questions would be possible.

```
SELECT *
FROM Person
WHERE Person.Description CONTAINS picture_ref

SELECT *
FROM Person
WHERE Person.Description PART_OF picture_ref

SELECT *
FROM Person
WHERE Person.Description LIKE picture_ref
```

4.4.3. External

External functions are functions that the CQL parser has no knowledge of. To the CQL parser external function calls are only a string that is passed down to the underlying DBMS.

```
SELECT *
FROM Project
WHERE EXTERNAL ExtFunc(Project.Video, filename, par1, par2)
```

5. References

[Bern93] M. Bern, P. Kool, P. Rosengren, U. Wingstedt, "Application Design with the Intuitive Tools - two case studies", SISU Report 1993:05.

[Bertino90] E. Bertino, F. Rabitti, "The Multos Query Language", in "Multimedia Office Filing - The Multos Approach", pp 53-74, ed. C. Thanos, North Holland, 1990.

[Celanto91] A. Celanto, M.G. Fugini, S. Pozzi, "Querying Office Systems about Document Roles", Proceedings of the Fourteenth Annual International ACM/SIGIR Conference on Research and Development in Information Retrieval, pp 183-189, 1991.

[Fuhr90] N. Fuhr, "A Probabilistic Framework for Vague Queries and Imprecise Information in Databases", pp 696-707, Proceedings of the 16th VLDB Conference, 1990.

[Fuhr93] N. Fuhr, "A Probabilistic Relational Model for the Integration of IR and Databases", pp 309-317, Proceedings of the Sixteenth Annual International ACM/SIGIR Conference on Research and Development in Information Retrieval, 1993.

[Motro86] A. Motro, "Baroque: A Browser for Relational Databases", ACM Transactions on Office Information Systems, vol. 4, No. 2, april 1986, pp 164-181.

[Rosengren93a] P. Rosengren, U. Wingstedt, M. Bern, P. Kool, "ER-Based Information Retrieval In a Mixed Database Environment", Proceedings of the 12:th International Conference of Entity Relationship Approach, 1993.

[Rosengren93b] P. Rosengren, U. Wingstedt, M. Bern, P. Kool, "A Tools Oriented Visual Interface for Multimedia Databases", NDA'93, International Symposium on Next Generation Database Systems and Their Applications, Japan September 1993.

[Rosengren93c] P. Rosengren, U. Wingstedt, P. Kool, M. Bern, "Accessing Information in Large Corporate Databases - The Intuitive Approach", SISU Report 1993:03.

[Rosengren93d] P. Rosengren, "Applications of a Multimedia Retrieval Information System - five case studies", SISU Document 10.

[Wingstedt93] U. Wingstedt, M. Bern, P. Kool, P. Rosengren, "Intuitive Tools for Information Retrieval - Requirements and Architecture", SISU Report 1993:04.

Appendix Syntax Description

Below we give a syntax description of CQL. A star (*) indicates repeating occurrences while square brackets indicates options.

```
/*-------------------------Query--------------------*/

<CQL-query>::=
<EntitySetDescription*> RELATIONSHIPS <RelationShipConstraint*>

<EntitySetDescription>::=
SELECT <Output>
FROM <EntityClassSpecification>
WHERE <Restrictions>

/*---------------------------OutPut--------------------*/

<Output> ::=
* | <OutputList> |

<OutputList> ::=
<AttrOutSpec> | <AttrOutSpec>, <OutputList>

<AttrOutSpec> ::=
<AttrClass> | SUM ( <AttrClass ) | MAX ( <AttrClass> ) | MIN (
<AttrClass> ) | COUNT ( * ) | COUNT ( <AttrClass> )

/*---------------------------Restrictions--------------------*/

<AttributeRestriction>::=
<AttrClassSpecification> <Operator> <Operand>
      | <AttributeRestriction> <BoolOp> <AttributeRestriction>
      | (<AttributeRestriction>)
      | NOT <AttributeRestriction>

<VagueRestriction> ::=
EntityClass . About ( number, <stringlist> )

<RelationShipRestriction> ::=
<EntityClassSpecification> <RelationShipSpecification>
<EntityClassSpecification>

<TextRes> ::=
TEXT <TextRestriction>

<TextRestriction>::=
      <TextSpecification>
      | <TextRestriction> <BoolOp> <TextRestriction>
      | (<TextRestriction>)
      | NOT <TextRestriction>
```

```
<ExternalRestriction> ::=
EXTERNAL function ( <AttrClassSpecification>, <Stringlist> )

<SimpleRestriction> ::=
<AttributeRestriction> |
<VagueRestriction> |
<ExternalRestriction> |
( <SimpleRestriction> )

<CompoundRestriction> :: =
<SimpleRestriction> <BoolOp> <CompoundRestriction> |
( <CompoundRestriction> )

/*---------------------------Specifications--------------------*/

<EntityClassSpecification> :: =
<EntityClass> | <Variable> [EntityClass*]

<RelationShipSpecification> :: =
<RelationShipClass> | <Variable> [RelationShipClass*]

<AttrClassSpecification> :: =
<AttrClass> | <Variable> [AttrClass*]

<TextSpecification> ::=
"string of characters" |
<string> WITHIN <ParSenWor> number |
<SenPara> <WordSpecification> <Position>|
ANY <WordList>

<WordSpecification> ::=
STEM <string>

/*---------------------------Keywords-------------------------*/

<ParSenWor> ::=
P | S | W

<Position> ::=
LAST | FIRST

<SenPara> ::=
SENTENCE | PARAGRAPH

/*---------------------------Operators------------------------*/

<Operator>: =
<ComparisonOp>| BETWEEN | IN | LIKE | CONTAINS | PART_OF

<BoolOp>::=
AND | OR
```

```
<Relop>::= <ComparisonOp> number

<ComparisonOp>::= <> | > | < | >= | <=

<Operand> ::=
<TextRes> |
<OperandList>

/*---------------------------Lists-------------------------*/

<OperandList> ::=
<NumberList> | <StringList> | <AttrList> | <FileList>

<NumberList> ::= number | number, <NumberList>

<StringList> ::= <String> | <String>, <StringList>

<AttrList> ::= <AttrName>| <AttrName>, <AttrList>

<Wordlist> ::= <WordSpecification> | <WordSpecification>, <Wordlist>

<FileList> ::= filename | filename, <FileList>

/*---------------------------Classes-------------------------*/

<AttrClass>::=
<EntityClass>. <AttrName>


<EntityClass>::=
string of characters

<RelationShipClass>::=
string of characters

/*---------------------------Names-------------------------*/

<AttrName> ::=
string of characters

<String> :: =
string of characters
```

SVENSKA INSTITUTET FÖR SYSTEMUTVECKLING
SISU

Electrum 212, 164 40 Kista
Isafjordsgatan 26
Telefon 08-752 16 00 Telefax 08-752 68 00