# Checking Feasible Completeness of Domain Models with Natural Language Queries

**Christian Kop**
Applied Informatics
Alpen-Adria Universitaet Klagenfurt
Universitaetsstrasse 65-67, 9202 Klagenfurt, Austria
christian.kop@aau.at

## Abstract

During the design of information systems it is important to know when a conceptual model of a database is complete. Completing the conceptual database model is a communication process between end users and designers. At the end of the process, both kinds of stakeholders agree that the model has reached a state where it can fulfill the purpose for which it is designed. Natural language queries play an important role as test cases in this process. They are understandable by the end user and help to discuss the model. Hence, this paper focuses on natural language queries as one mean (among others) to test if a conceptual model (domain model) is complete.

*Keywords:* domain models**,** controlled natural language queries, completensess, quality, end user participation

## 1 Introduction

In information systems and data centric applications it is necessary to check if all the necessary concepts are modeled. Therefore, such a domain model must be continuously checked in order to be complete. Completion is a communication process between end users and designers. At the end of the process, both kinds of stakeholders agree that the model has reached a state that fulfills its purpose. Speaking in terms of feasible completeness introduced by Lindland and Solvberg et al. (1994) they stop the completion process since "*further modeling is less beneficial than applying the model in its current state*" (Lindland and Solvberg et al. (1994), p. 46). For instance, a class diagram or entity relationship diagram, which is used later on for a database can be seen as complete if

- The relevant classes (entity types) attributes and associations (relationships) appear in the model.
- Each attribute has the necessary data type
- The multiplicities are specified on each association
- The multiplicities of multi-valued attributes are specified
- If necessary, a default value is specified for an attribute
- If necessary, it is specified if the value of an attribute is mandatory or optional
- If necessary, attributes that have unique values are declared.
- If necessary, it is specified if the value of an attribute follows a certain format (e.g., format of a string that must be a valid e-mail address).
- There is no open question or open task related to a modeled attribute or class.

Some of these listed items are interesting for both, the end user and conceptual modeler (i.e., classes, attributes, associations). Some are more interesting for conceptual modelers (e.g., the right data type, optional and mandatory attribute values). The modeler might be also interested in progress aspects (e.g., annotating open questions to model elements).

Different presentation techniques can be used to find defects and incompleteness of models. In Kop (2009a) some of them are discussed. However, those techniques depend that some information in the model already exists. What about a complete new concept, which actually does not yet appear in the model. In other words, this paper focuses on the question: *Are really all relevant concepts already collected?*

The purpose of the model must also be considered. Usually, a conceptual database model is developed to describe structural aspects. This structure also determines the possibilities of the retrieval of data. The earlier, this purpose can be tested the better it is. Therefore an additional focus is: *Does the model structurally support retrieval of data?*

For these questions, an additional, purpose driven technique is needed. That means, a new concept is only added, if it supports the retrieval purpose.

At least one researcher gave an interesting manual solution how this can be achieved. In one of the exercises given in his book, Rumbaugh (Rumbaugh 1991, p. 193) explicitly gives an object diagram for a sports scoring system and asks the reader to check manually if a couple of natural language queries can be applied on this object diagram.

Also a practical experience underlined the necessity of such an approach. Requirements were collected in several workshops. In one of the workshops the end users were also asked to give examples for queries, which will be applied on the database under development. They were very engaged to produce such queries.

Although the initial idea of manually comparing natural language queries with a domain model was found in the OMT book (Rumbaugh 1991), no research was found so far, which describes in detail how such a computer supported approach can look like. Therefore, this paper addresses this research topic.

Tagging and chunking will be used as the linguistic instruments and basis for natural language query analysis. Furthermore, it will be described, which additional information a meta-model must provide for this task.

The paper is therefore structured as follows. Section 2 gives an overview of work, which is related to this topic. In this section also an overview of the previous work will be given. Section 3 shows the aim of the approach by giving a motivating example. Section 4 describes how the meta-model must be extended, to be itself a support in this process. Section 5 describes details on tagging and chunking for natural language queries. Section 6 focuses on the tests of the query analyzer and describes the tool. It also discusses tagging and chunking for analyzing natural language queries. Section 7 presents conclusions and future work.

## 2 Related Work

### 2.1 Manual Strategy

In his book on Object Oriented Modeling, Rumbaugh et al. gave an exercise (see p. 193). He provides the reader with a partly completed object model in the sports domain. In addition, the exercise contains 12 queries (e.g., “Find all the members of a given team”, “Find the net score of a competitor for a given figure at a given meet” etc.). In this exercise, the reader is asked to explain how the object diagram can be used for each of the specified natural language queries. The exercise in the book is continued and the queries are even reused. The reader is asked to add methods to the object diagram to satisfy the queries (Rumbaugh 191, p. 294). This book gives a good hint for using queries as test cases at an early stage of modeling but the procedure must be done manually. No computer support is given, which tries to match the notions in the query with the concepts in the object diagram.

### 2.2 Querying an Existing Database

Much research has been done to query existing databases. In these approaches the final and stable database is implemented and filled with data. Techniques of natural language querying are described in many research works (Berger et al. 2003, Hofstede et al. 1996, Kardovácz 2005, Kapetainos et al. 2005, Kao et al. 1988, Owei and Navathe et al. 1996, Stratica et al. 2005, Meng and Siu, 2002, as well as Satori and Palmonari 2010). Some operate on a relational database, others on a conceptual model of the existing database. Some use additional information derived from linguistic lexicons or ontologies. For instance, such ontologies are needed to expand the notions found in queries to similar notions (synonyms etc.). This step is necessary in such approaches, since it cannot be assumed that a query notion matches with an identical concept of the database. Machine learning approaches for natural language query parsing were used by Mooney et al. (Tang 2001, Ge 2005, Kate 2006, Wong 2006) in the Geo Query Project.

Panchenko et al. (2011) introduced an approach that uses controlled natural language queries for querying source code elements in a source code repository.

Visual query tools, were described in the following works (Bloesch et al. 1996, Owei and Navathe 2001, Jaakola 2003, Järvelin et al 2000). These tools operate on the conceptual model and their main purpose is to produce SQL statements, In these tools, needed classes and attributes are graphically selected. Beside these, Embley (1989) introduced forms to generate queries. This strategy is also proposed by Terwillinger and Delcambre et al. (2007).

The main objective of all techniques is to support either the generation of a SQL query that can be executed on the relational database or to directly retrieve data from the database. As mentioned before, this implies a database filled with data. It is thus not the task of these techniques to check if something in the model is missing.

Opposite to that, the approach mentioned in this paper is applied during domain modeling as early as possible. Therefore, the focus is not the generation of SQL but to test if the model is complete. It is thought as a help, whenever it is necessary to check the model, before any prototype or user interface form is built. Opposite to the graphical query languages where the query is constructed by navigating through the model, the end user must not see the model during the creation of the queries. This has the advantage that the user is not influenced by the model but freely names the notions, which he needs for the query. The query then helps to check if the designed model can handle the notions used in the query.

### 2.3 Test Driven Development

Test driven Development, which is proposed by Beck (2004) is included in this related work section, since its paradigm can also be applied in this approach. In Test Driven Development, the developer is enforced to design and write test cases for small sized problems before he starts to design and implement parts of a system. After he has generated these test cases, he writes the first implementation of the requirements, which he lets fail. Iteratively, he improves his implementation until it is successful for the written tests. The paradigm behind this is pointed out by Kent Beck: “*Failure is progress*” (Beck 2004, p. 5).

Using natural language queries as test cases can be similar to this paradigm, since there is no need for a nearly complete conceptual model of the database, which has to be implemented. The process of improving the conceptual model can also start with a very lean initial model. During the “tests”, the model iteratively grows and gets improved. As a side effect, the stakeholders learn more and more about the model.

### 2.4 Model Quality Approaches

According to Lindland and Solvberg et al. (1994) three dimensions have to be considered for conceptual modeling quality, namely: syntax, semantics and pragmatics. If the model follows the rules and the grammar defined in its corresponding meta-model, then the model has a syntactic quality. Semantic quality is given; if the model only contains true statements of the domain and is complete (no important concepts or statements are missing). Lastly, pragmatic quality relates the model to the interpretation of the user. A pragmatic quality of a model is given, if it is understandable to a human stakeholder or a system, which has to understand the model. Since quality of a model in general and in particular completeness of a model cannot be fully achieved, the authors have introduced the notions feasible validity, feasible completeness (semantic quality) and feasible comprehension (pragmatic quality). That means, that the model can be improved "*until it reaches a state, where modeling is less beneficial than applying the model in its current state*" (see: Lindland and Solvberg et.al, p. 46, (1994)).

In order to improve the quality, Assenova and Johannesson (1996) propose to use well defined model transformations. Moody (1996) describes that comprehensibility of a conceptual model can be enhanced if icons and pictures are introduced instead of simple graphical primitives (rectangles, ellipses etc.). In an extensive literature study of the same author (Moodey 2005) on the quality of model, it was concluded that conceptual modeling must shift from an art to an engineering discipline where quality plays an important role.

Easterbrook et al. (2005) made an exploratory study and showed that view points are also an important technique to improve the model quality.

The verbalization approaches described by Dalianis (1992) and Halpin et al. (2006) aim at getting a better understanding of the conceptual model by presenting users a natural language translation (verbalization) of the model.

Batini, Ceri and Navathe (1992) used Dataflow diagrams to complement and complete the conceptual models.

The approach described in this paper also aims to improve the completeness of a conceptual model. Therefore it can be seen as an additional support for checking the quality of a domain model.

## 2.5 Previous Work

In the own previous work (Kop 2009a) visualization techniques (graphical visualization, tabular representation and verbalization) were discussed. It was also described how some of them can help to detect incompleteness. For instance, a cell, which is empty in a tabular representation is a hint that something in the model is missing. Model elements can be annotated with explicit progress information (e.g., an open question for a certain model element). Such annotated elements give hints for model incompleteness. However, all these techniques present incompleteness on the basis of existing model elements. This supports the extension of the model elements but it does not guarantee that no important concept is forgotten. Therefore, in another previous work, (Kop 2009b) controlled natural language query pattern and a parsing approach is introduced. A Controlled Natural Language is a language that has a restricted grammar and dictionaries (Fuchs et al. 2005). Though, first results were promising, it turned out that different sentence patterns can contradict each other if the language grows.

Whereas previous work focused visualization techniques, parsing and the communication process, this paper focuses on query analyzing based on tagging and chunking and a supportive meta-model.

## 3 A Motivating Example and Aims

The main aim is to check if each notion, which appears in a query is also represented in the model. Otherwise the model must be refined. Beside this, such an approach can support additional aims, which will be described in this section.

To show how the approach works, it will be motivated by the example domain taken and adopted from the Geo Query Project. Whereas there, the database model was a basis for querying the database and a machine learning approach for natural language query parsing, here it is used as a motivating example that queries can also be applied on a work in progress model. Therefore, it is supposed that designers and end users still work on the model presented in Figure 1. In this figure a first intermediate result is presented, which should describe some information about the states in the US.

The model is not yet finished. The stakeholders now decide to use queries to see if something is missing.

Now it is supposed that the end users would like to use the query "Which mountains exist in the several states" in the final database, which is currently represented only by the intermediate work in progress model of Figure 1. As it can be easily seen, there is already a concept "state" in the model but no concept "mountain" can be found. Such a query won't work in the final database if the model is not refined and the concept "mountain" as well as all its related information (i.e., attributes and associations to other classes) is added to the model. This example demonstrates the first important aim of the approach.

The queries support to find gaps in the model.
This is also a step towards Kent Beck's statement "Failure is Progress", since the stakeholders know that something is missing and they must refine the model.

However, the idea to apply queries on a model can have other advantages even if the model is (nearly) complete. Now it is supposed, that the following query is applied on the model: "Which rivers flow through the US". The notion "rivers" is found in the model. Therefore, it can be said that there is at least minimal information provided by the model, which can help the query to become successful.

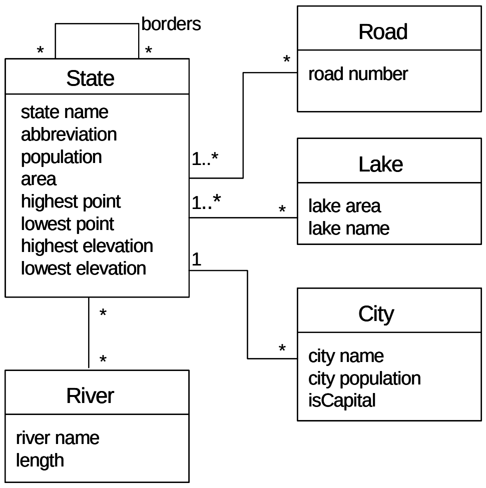


**Figure 1: Example of a Domain Model**

Of course, if the concept found in the model is a class (e.g., “river”), then the stakeholders must be aware, which kind of information (all information about rivers or only specific information – e.g., the river name) is needed. Another thing that has to be considered is the notion “US”. A query applied on a work in progress model can support the detection of more information than model concepts. If the model is build for all nations in the world and state is synonymously used, then “US” is an example of a state. Particularly, it might be the value of the “state name”. However, if the model only represents the information about US states as it is intended here, then “US” can be treated more like a view (i.e., “all states”). Such a view might become an external database view. Once again “failure” (i.e., “US” is not found in the model the first time) can be a progress. After applying the query on the model, the stakeholders know that in some queries a view on all the states is required. The notion “US” can now be collected and related to specific concepts in the model. The next time, a query contains the notion “US” the tool knows that “US” must be replaced by its related model concepts and such a future query will be successful. Of course it should be clear, that the model must be refined if neither “US” nor “state” appear somehow in the model. In such a case the stakeholder once again have the progress information that the current model needs completion.

Even if a query is successful (i.e., nothing is missing) this can be an important progress information. The stakeholders know that the model is at least complete with respect to this query.

All this shows further aims of the approach. With queries, the detection of synonyms (another, linguistic view on a concept), external database views and examples can be supported. All this information is very useful in further stages of information systems development (e.g., examples can be used for testing the system after it has been implemented).

Finally, the queries itself can be reused later on. This is another big advantage and it can happen in two ways:

- The queries are functional requirements of how the end user wants to retrieve the final database. Queries or methods (Rumbaugh 1991) can be developed manually from them.
- In more advanced information systems (see Related Work Section), queries and the additional information (i.e., synonyms, external views) can be the basis for a natural language query engine, which automatically retrieves the data from the database.

To summarize the aims: Something, which looks like a failure at the beginning, is a progress in a project.

Particularly, the approach helps to find gaps (i.e., missing concepts and associations); detect synonyms, external views and examples; get functional requirements; and to get a basis for a natural language query engine.

## 4 Metamodel, Process and Decisions

If query notions, which are not directly used as model concepts are collected for future checks, then the meta-model must be extended to give support. The next section explains the meta-model and the ideas behind it. Afterwards, the process is briefly described. Finally, the decisions, which must be made by the stakeholders are described.

### 4.1 Meta-model

Usually, each data model consists of classes, attributes, associations (relationships) and other notions. Figure 2 shows the excerpt of the meta-model. It focuses on the classes, attributes and the surrounding notions, which are necessary to fulfill the aims of the approach. Each class and attribute, which appears in the conceptual data model, is generalized to the notion concept. Hence, one important goal of a notion found in a query is to match it with a concept in the conceptual data model. If this fails, this can be a hint to refine the model. However, the notion in the query can also be a circumscription of one concept or many concepts used in the data model. In its simplest form, the extracted query notion is a synonym of the concept. To be able to handle this in further queries such terms are collected in “synonym” (see Figure 2 – meta-model). For instance, city and town might be synonymously used in a certain domain. A notion in the query can also be a view on one concept. For instance, in the Geo Query Corpus, “US” is often mentioned in several queries. However, “US” is not an example of one of the data model concepts. Instead “US” means: “All states”. Hence, it is a specialized view, which is based on the main concept “state”. “US” is therefore collected in the concept view descriptor (see meta-model in Figure 2). It is related to the concept “state” via the has-main-concept relationship. Whenever “US” is mentioned in a new query, then it can be seen as a view on states. No refinement is necessary, since it can be shown, that the data model contains this notion. Other examples for concept view descriptors in a university domain are “good student” or “good mark” respectively. Let’s suppose the model for the university already contains the concepts “student”, “course” and “mark”. If “good student” is not found in the model itself but can be resolved by a phrase “A good student is a student, which has only marks A or B in courses”, then “good student” is stored as a concept view descriptor. Student itself is the main concept. The subordinated concepts are “mark” and “course”. A definition of a good student as given above is stored in the concept view definition.

If a notion like “good mark” appears in the query and it can be described as “good mark is A or B” then mark is the main concept. No subordinated concepts exist here, since the main concept is an attribute, which is defined by its specific values. This strategy is also applied, if the concept view descriptor does not directly match with a concept in the model. Let’s suppose that the query has the notion “good grade”. If “grade” itself can be treated as a synonym of “mark”, then “good grade” is simply stored in the concept view descriptor.

Finally, if a notion is mentioned, which covers more than one concept it is stored in the model view descriptor. A model view descriptor is a specialized view on several co-equal model concepts. An example would be, if the query asks for “List the shipments” and shipment is already realized as a relationship between “product” and “customer” in the existing model (i.e., products are shipped to customers). Another example in a university domain would be “List the academic staff”. If academic staff must not be modeled as an own class in the domain model but is only a view on the union of already given model concepts “professor” and “assistant”, then it is stored in the model view descriptor. The definition of the view is stored in the definition property of the model view descriptor. The relationship has-involved-concepts (see meta-model in Figure 2) relates “academic staff” with “professor” and “assistant”.

Another special kind of view is the aggregation view. Most often maximum or minimum values are needed (e.g., “the longest river”). In such a case, the notion “longest river” is stored as an aggregation view descriptor. In the definition this descriptor is replaced by a maximum- or minimum function and this function must be applied on an attribute (e.g., “max(river.length)”). If the aggregation view descriptor “longest river” is mentioned a second time in another query, then it can be resolved with the aggregation function of the attribute.

If an instance or value is mentioned in the query, then this instance can be stored in examples and related to the concept to which it belongs. The next time the example is mentioned, it is correctly replaced by its concept. It is possible to store value descriptors (e.g., “large”, “old”) if these descriptors can be extracted from a query clause (e.g., “is old”, “must be old” etc.).

Of course, in all cases, in which a concept in the model cannot be found, it must be checked if the model can be refined.

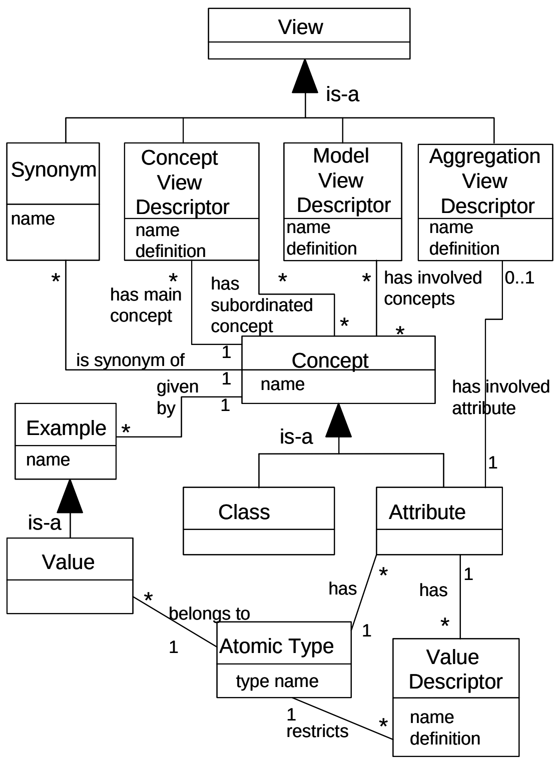


Figure 2: The Meta-model

## 4.2 Process

The communication and negotiation process starts with an initial model. This model is derived from requirements. Afterwards, the stakeholders (i.e., conceptual modelers and end users) have to identify and generate the queries, which will be applied on the work in progress model. In the optimal case, nobody of the stakeholders should see the initial model. They should use their own words. This shall also help to produce views of the concepts and the model. Remember, according to Kent Beck’s position “Failure is Progress”, progress information can be found. Afterwards, the queries are executed automatically on the current model. The tool and its linguistic instruments, which will be described in Section 5, are responsible for this task.

Based on the reports the tool produces, the stakeholders must discuss the model. If failures are found then new domain model requirements can be derived from them. Decisions, which will be described in the next subsection, must be made. The decisions have an influence on the refinement and completion process of the model. This whole process is finished if all the stakeholders agree that no more new information can be found or are required.

## 4.3 Decisions

If notions extracted from the queries do not match, it is the task of the stakeholders to decide how these query notions can be related to the model. Particularly, the stakeholders have to decide the following:

- Is it a missing concept of the model? In that case the model has to be refined and the missing concept must be integrated into the model. Integration means, that a class must be related to other classes with associations, an attribute must be related to the class to which it belongs.
- Is it an example of a concept, which is missing? In that case the model has to be refined and the missing concept must be integrated into the model. Furthermore, the example must be collected and related to the integrated concept.
- Is it a value descriptor but the concept (attribute) to which the descriptor belongs does not exist? In this case, the model has to be refined. At least the missing attribute (may be, also the class of the attribute) must be added. Then, the descriptor and the examples, which are subsumed by the descriptor, must be added.
- Is it a view but the concepts, on which the view is based, do not currently exist? In this case, the missing concepts must be integrated. The view must then be collected for these concepts.
- Is it an example of a concept, value descriptor or a view and all the necessary concepts already appear in the model? In this case, only the example, value descriptor or the view respectively have to be collected and related to corresponding concepts.

## 5 Tool Architecture, Linguistic Instruments and Query Interpretation

According to the last section, in which the meta-model is explained, the most essential task that a tool must provide is the extraction of noun phrases. These noun phrases can then be matched against existing concepts in the model or related notions, which describe views or examples of model concepts. It is not only necessary to extract simple words (e.g., “person”). If there are compound nouns (e.g., “lake name”) or if there are nouns that are specialized by adjectives (e.g., “highest elevation”) then these words must be treated as a single notion. Well known linguistic instruments for achieving this are tagging and chunking. Generally speaking, tagging is the basis that relates words to categories (e.g., lake = noun) but a word like “lake name” or “highest elevation” is still treated as a sequence of independent tagged (categorized) words. Chunking can be built upon tagging. It examines the tagged word sequence and summarizes certain sequences to a higher level element (e.g., a noun phrase). In the succeeding subsections, firstly an overview of the tool architecture is given. Afterwards, the basic linguistic instruments, namely tagging and chunking in the context of natural language queries, will be described in more detail. Finally, it will be explained how the chunking output is used in the query interpreter and what the purpose of the matching module is.

## 5.1 Architecture

The query analyzer consists of a graphical user interface (GUI) for interaction with users. The query text, which is loaded from a file or entered in a text area field is forwarded to the query interpreter, which itself hand the text over to supporting modules (chunking and tagging). The most basic linguistic module is the tagging module. It analyzes the text and returns the output to the chunker. The chunker subsumes the results from the tagger and forwards it to the query interpreter. This module then extracts the necessary notions. The notions are forwarded to a matching module, which interacts with the stored work in progress domain model. The matching module tries to compare the extracted notions with notions already collected for the model (i.e., the model concepts itself as well as views and examples). Figure 3 shows the architecture.

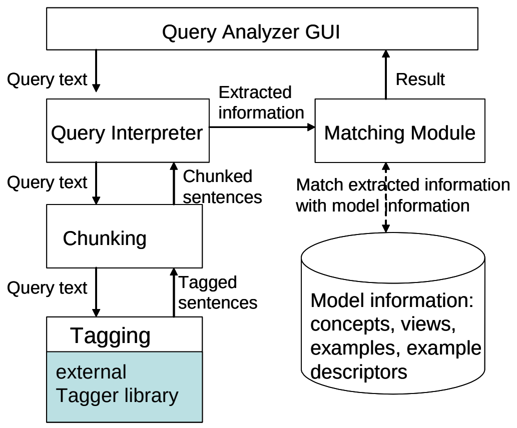


**Figure 3: Architecture of the Prototype**

## 5.2 Tagging

For the initial step of linguistic analysis the Stanford Tagger is used (Toutanova et al. 2003). A tagger is a tool, which takes as input a text and returns a list of sentences with tagged words (i.e., words categorized as noun, verb, adjective etc.). The chosen tagger categorizes the words according to the Penn-Treebank TagSet. In this tagset the word categories together with some important linguistic features of a word are encoded. If a noun is in plural then the category NNS is chosen. If a proper noun is detected then NNP is used.

For instance, for the query “which mountains exist in the several states”, the external Stanford tagger library produces the following output: “`Which_WDT mountains_NNS exist_VBP in_IN the_DT several_JJ states_NNS`“**.** At the end of each word token, the tagger adds the category (e.g., WDT = interrogative determiner/wh-determiner; NNS = plural noun, VBP = verb non 3rd person form; IN = preposition or subordinating conjunction; DT = determiner, JJ = adjective)

The external tagger library is enapsulated in a tagger module. In this module the output is checked for inconsistent word categories. If a wrongly categorized word is found, it is changed to another category that better fits with the context in which it is embedded. The other task of this additional module is to prepare the tags for the chunker module. For the given example query, it is checked if JJ has a more specific meaning. Words like “several”, “many”, “much” are categorized as adjectives but they do not specify a characteristic of a concept. In fact, they tell something about the set of concepts. For such words, in the linguistic approach NTMS (naturalness theoretic morphosyntax) developed by Mayerthaler, Fliedl and Winkler (1998) the linguistic term quantifier was introduced. Here, the term quantifier was adopted. Thus, the category of the word “several” is changed from an adjective to a quantifier.

## 5.3 Chunking

Chunking is useful to group words to a chunk that can be seen as a phrase (e.g., a verb phrase or a noun phrase). Details of chunking are described e.g., in (Sang and Buchholz 2000). The implemented chunker, provides two steps.

In the first step, it tries to find noun phrases and verb phrases. The chunker module clusters nouns (e.g., *customer number*) as well as word categories strongly related to nouns (e.g., articles, adjectives, quantifiers) to a noun phrase (e.g., *the customer number*). In general, the implemented rules follow the common rules of chunking (e.g., article + adjective + noun = noun phrase). It subsumes verbs and word categories, which are strongly related to them (e.g., adverb, verb particle) to a verb phrase. One exception exists. If the words “many” or “much” follow the word “how” (e.g., “how many persons”) then a word like “many” is not chunked with “person” to a noun phrase but it is grouped with “how”. Hence, instead of the output [how] [many persons] the query chunker generates the output [How many] [persons].

In the example “Which mountains exist in the several states“, the chunker module generates the noun phrase chunks NP [mountains] and NP [the several states]. It also generates a verb phrase chunk, but this chunk only contains VP [exist]. Most of other tags are taken as they are. In the case of a preposition, the name P was chosen. Hence, an abstraction of the result of this step can be presented as WDT [Which] NP [mountains] VP [exist] P [in] NP [the several states].

In the second chunking step it clusters a list of noun phrases separated by prepositions to a more complex noun phrase (CNP). This result is then forwarded to the query interpretation module. An abstraction of the result can be presented as CNP [NP [mountains]] and CNP [NP [the several states]]. This step is intended for phrases like "… the several states of the US". Here the abstracted result is: CNP [NP [the several states], PP [P[of] , NP [the US]]]. The reason is, that the phrase "… the several states of the US" is one noun phrase.

## 5.4 Query Interpretation

The last linguistic step is the interpretation of the chunker output. It is a combination of noun phrase extraction and more refined parsing of specific patterns.

In a first step the query interpreter extracts all the noun phrases. This guarantees that at least query notions can be extracted even if a more specific pattern cannot be detected. The found query notions are used to check if they match against existing concepts, views or examples (see Subsection 5.5). In our example about the mountains and the several states, the query interpretation module examines each complex noun phrase (CNP). It iterates through all the noun phrases within a CNP and extracts the nouns. If an adjective modifies a noun, then both the adjective and noun is extracted. However, if a quantifier like several, many etc. modifies the noun, then only the noun is extracted. Determiners (e.g., "the", "a", "an") are ignored for the extraction process. Also the first noun phrase is ignored if the first noun phrase only contains meta- information (e.g., "set", "list" in phrases like "… the set of …", "… the list of …").

More specific patterns are constraint sentences (e.g., "The height must be greater than 3000"). These sentences can be used within a query text to constrain the query itself (e.g., "Tell me the mountains in Colorado"). Such constraint sentences can also have adjectives at the end (e.g. "must be old"). If such adjectives are found, then these adjectives are collected as value descriptor candidates. Other, more specific patterns are values or numbers at the end of a noun or proper nouns following the verb "is".

## 5.5 Matching and Path Finding Module

If all the notions are extracted the system tries to match the notions found in the query with the concepts, views and examples actually collected for the domain model. If all the notions in the queries are found in the model or in model related information (i.e., views, examples), then the query is successful. To achieve this, the extracted notions are firstly compared with the concepts in the model (i.e., can the extracted notion, or its singular form be found in the model). If an exact match is not successful, then the modul tries to match the head of a noun phrase. For instance, if the module cannot find a match for an adjective+noun combination (e.g., "good student") then it tries to find a match for the noun only. If this does not work then the extracted notion is searched in the views (synonym, concept view descriptor or model view descriptor). If this doesn't work either, the examples are examined (i.e., is the query notion an example of a concept). Since the views as well as the examples are related to a model concept, this notion can be traced back. Therefore, in any of the above mentioned cases, the notion found in the query can be replaced by the concept in the model to accomplish the next step. If all the notions extracted from the query are found in the model, then the tool can determine a path between these model concepts.

Path finding is done by checking if all the concepts, which are necessary for the query belong to the same connected component within the conceptual model graph. With this step it is possible to find missing associations.

## 6 Tests and Prototype

### 6.1 Tests

Natural language queries, which were found in literature and own created queries were taken as test cases to test and improve the linguistic instruments. Among these test cases, the greatest set of natural language queries came from the Geo Query Project. In this project 880 query sentences are used. Theses sentences can be categorized in queries starting with "What", "How", "Which", "Where" and other queries. These other queries do not start with an interrogative but start with a verb (e.g., "list", "give me", "name the", etc.) or they neither start with a verb nor with an interrogative (e.g., only a noun phrase is used for the query). The majority of query sentences is provided for queries starting with "What" followed by "How" and "Which".

Especially in English, words often can be used as a noun and a verb. This can cause a wrong categorization, since the tagger assumes a noun although a verb is needed. For instance the word "border" can be at least categorized as a noun (e.g., "the border") or a verb (e.g., "to border"). In an incomplete domain model you cannot rely that the model itself can disambiguate the word to a specific category (e.g., either verb or noun). Another example is "name the capital … ". The tagger treats name as a noun. However, in the context of the query "Name the …", the sentence starts with a verb. Therefore, some cases would fail because the tagger wrongly categorizes a word in the sentence. With the additional consistency checking step within the tagging module, such problems can be considered. This was done by introducing an additional context window (sequence of word categories), in which elements of that sequence are compared with their neighbors. Also some hard problem cases were detected, in which a solution could only be achieved by broadening the context window. Such a problem appeared for instance with "which" in the middle of a sentence. A query like "Which person offers which course" is only analyzable, if the whole query is treated as the context.

In the Geo Query Corpus proper nouns were written in lower characters (e.g., "texas" instead of "Texas"). This happened, since the queries were extracted from a knowledge base where all words of a query must be written in lower case. In most of the cases this is not problematic, since the tagger categorizes such a proper noun as a common noun. This is not perfect, but at least it had the effect that they could be extracted as relevant query notions. In some specific cases however, tagging even did not classify such words as nouns. This problem can only be fixed if the writer is enforced to write proper nouns correctly with a capital letter at the beginning.

To give the reader an impression what kind of syntactical structures of query sentences can be analyzed, a further example of the Geo Query Project is given and some examples from other literature.

One of the complex queries in the Geo Query Project is: "*What are the major cities in the states through which the major river in Virginia runs.*" The query interpreter extracts "major cities", "states", "major river", "Virginia". Afterwards this information is matched with the model and model related information (e.g., examples, views).

The work of (Owei and Navathe 2001) gives the good query example "*What courses is the student whose name is Marshall taking from associate professor 'Jones'.*" The query interpreter extracts "courses", "name", "Marshall", and "associate professor 'Jones'". Furthermore, it is extracted that "associate professor is restricted by "Jones" and "name is restricted by Marshall". The extracted two constraints help to indicated that an attribute is needed if, for instance, "associate professor" is already modeled as a class.

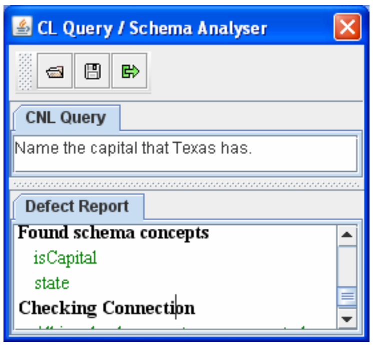
Other query sentences similar to the Geo Query sentences but related to the traveling domain were found in (Meng and Siu, 2002). One of these sentences is: "*Give me the least expensive first class round trip ticket on US air from Cleveland to Miami.*". Here the extracted query notions are "least expensive first class round trip ticket", "US air", "Cleveland" and "Miami".

Finally another query sentence, which Rumbaugh et al (1991) gave in the exercise within the OMT book is: "*Find the set of all individuals who competed in all of the events held in a season.*". Here the notions "individuals", "event" and "seasons" are extracted by the interpreter.

It can be concluded, that tagging and chunking provides more flexibility for sentences than parsing can do. With tagging and chunking it is at least possible to extract important notions though the whole syntactic structure cannot be completely parsed. These extracted notions can then be matched with elements of the work in progress domain model. However, even with tagging and chunking it turned out, that too much syntactical freedom is not successful and regulations must be introduced. The examples given above should give an impression about possible syntactical variations of natural language queries. Since the queries here are treated as functional requirements, regulations can be introduced with the same purpose as in the field of requirements engineering (i.e., to avoid misinterpretations).

## 6.2 Prototype

The prototype is implemented in Java. The query analyzer tool is an add-on to an existing tool, which graphically represents classes and attributes as nodes. Associations between classes appear as edges between two class nodes. An edge between an attribute and a class indicates that the attribute belongs to that class. The query tool itself has a text area input. It also accepts more than one sentence (e.g., "Name the river lengths in Colorado. The length must be greater than … "). If the query is successfully applied on the model, the relevant nodes and edges are highlighted such that the user sees the path between these nodes. In the tool this is done by painting the relevant edges and nodes with red color. However, if a notion in the query is not found in the model (neither as a concept nor as a view or example) then an error is presented to the end user. The next four figures show two examples. In the first example the query "Name the capital that Texas has" is successful, since capital can be found in the model and Texas was already collected as an example of the model concept "state". Therefore, in the editor view the path between the found concepts is provided to the end user. The query analyzer GUI prints no errors (see Figures 4 and 5). In the second example (e.g., "Which mountains exist in the several states") the notion "mountains" is not found in the model or in the related views and examples. Therefore, the query analyzer GUI returns an error message. Only the found concept "state" is highlighted in red in the graphical view (see Figures 6 and 7).


CL Query / Schema Analyser

CNL Query

Name the capital that Texas has.

Defect Report

Found schema concepts
isCapital
state
Checking Connection

Figure 4: Report for the first query

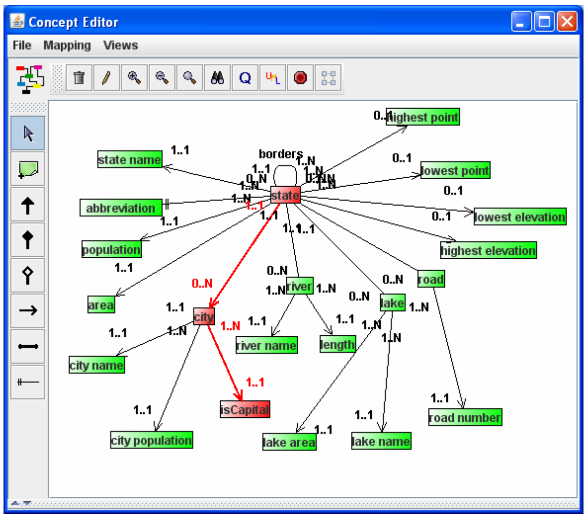
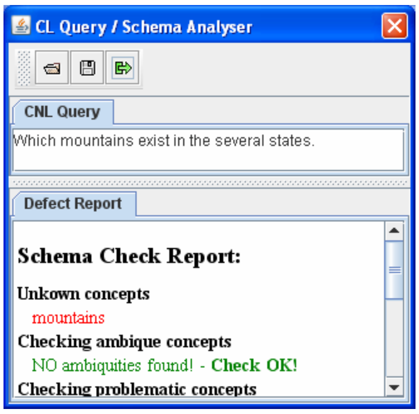


Figure 5: Graphical result for the first query


CL Query / Schema Analyser

CNL Query

Which mountains exist in the several states.

Defect Report

**Schema Check Report:**

Unkown concepts
mountains
Checking ambique concepts
NO ambiquities found! - Check OK!
Checking problematic concepts

Figure 6: Report for the second query

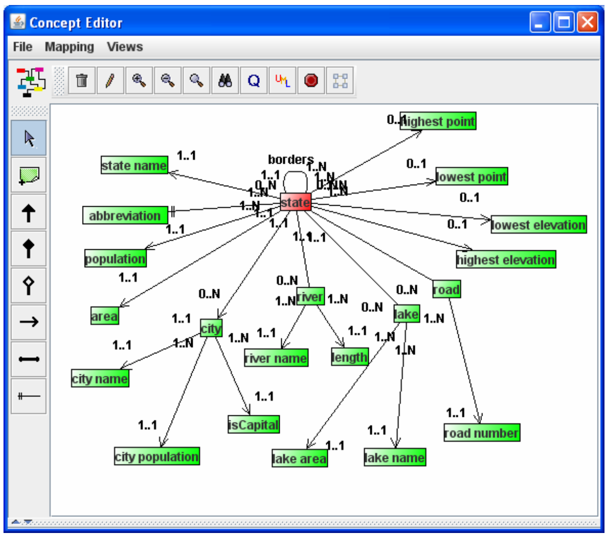


**Figure 7: Graphical result for the second query**

## 7 Conclusion and Future Work

It is clear that full completeness cannot be achieved. Instead, feasible completeness is the goal of a completion process. A mix of techniques, each supporting a certain aspect must be used. In this paper one possible approach within this technique mix was presented.

Particularly, it was explained how natural language queries as test cases for a domain model can be supported by a meta-model and the linguistic instruments tagging and chunking. From the tests it can be concluded that tagging and chunking is helpful. However, regulations are still needed to avoid misinterpretation. Hence, it remains still a controlled natural language. Especially, this is necessary if much more information than only noun phrases must be extracted. For instance if a query should check if two concepts are directly related to each other or if multiplicities are specified, then a restrictive sentence pattern is necessary (e.g., Is X related/connected to Y, Does X own at least 1 Y). As it is exemplified in requirements engineering such regulation are not only restrictions but can also be seen as a support for the stakeholders to define clear and comprehensible query requirements.

Future work will continue to collect further query test cases and to apply further tests on the query analyzer.

More attention will also be given to the association names and their roles in the process of completing the model with natural language queries.

**Acknowledgement:** I thank the persons, who were involved in the review process of this paper. Their hints and suggestions were very helpful for the improvement of the paper.

## 8 References

Assenova P. and Johannesson P. (1996): Improving the Quality in Conceptual Modelling by the Use of Schema Transformations. *Proceedings of the 15th International Conference on Conceptual Modeling*, Cottbus, Germany, LNCS **1157**, 277 – 291, Springer Verlag.

Batini, C. Ceri, S., and Navathe, S. (1992): *Conceptual Database Design – An Entity Relationship Approach*. The Benjamin Cummings Publishing Company.

Beck, K. (2004): *Test Driven Development by Example*. Addison Wesley Publishing Company, 5th Printing.

Berger, H., Dittenbach, M., and Merkl, D. (2003): Querying Tourism Information Systems in Natural Language. *Information Systems Technology and its Applications – Proceedings of the 2nd Conference ISTA 2003*, GI Lecture Notes in Informatics, **30**, 153 – 165, Koellen Verlag, Bonn.

Bloesch, A.C. and Halpin, T.A. (1996): ConQuer: A Conceptual Query Language. *Proceedings of the 15th International Conference on Conceptual Modeling*, Cottbus, Germany, LNCS **1157**, 121 – 133, Springer Verlag.

Controlled Natural Language: http://sites.google.com/site/controllednaturallanguag e/. Accessed 13. Oct. 2011.

Dalianis, H. (1992): A method for validating a conceptual model by natural language discourse generation. *Proceedings of the Fourth International Conference CAiSE'92 on Advanced Information Systems Engineering*, Lecture Notes in Computer Sciences LNCS **594**, 425 – 444, Springer Verlag.

Embley, D.W. (1989): NFQL: The Natural Forms Query Language. *ACM Transactions on Database Systems*, **14**(2), 168 – 211.

Easterbrook, St., Yu, E., Aranda, J., Fan, Y., Horkoff, J., Leica, M., and Quadir, R.A. (2005): Do Viewpoints Lead to Better Conceptual Models? An Exporatory Case Study. *Proceedings of the 13th IEEE Conferences n Requirements Engineering (RE'05)*. 199 – 208, IEEE Press.

Fuchs, N.E., Höfler, S., Kaljurand, K., Rinaldi, F., and Schneider, G. (2005): Attempto Controlled English: A Knowledge Representation Languagem Readable by Humans and Machines. *First International Summer School 2005, Lecture Notes in Computer Science* LNCS. **3564**, 213 – 250, Springer Verlag,

Ge, R. and Mooney, R.J. (2005): A Statistical Semantic Parser that Integrates Syntax and Semantics. *Proceedings of the Ninth Conference on Computational Natural Language Learning*, Ann Arbor, MI, 9 - 16.

Geo Query Project http://www.cs.utexas.edu/users/ml/geo.html. Accessed 13. Oct. 2011.

Halpin T. and Curland, M. (2006): Automated Verbalization for ORM 2. *Proceedings of OTM 2006 Workshop -On the Move to Meaningful Internet Systems 2006*, LNCS **4278**, 1181 – 1190, Springer Verlag.

ter Hofstede, A.H.M., Proper, H.A., and van der Weide, Th. P. 1996: Exploring Fact Verbalizations for Conceptual Query Formulation. *Proceedings of the Second International Workshop on Applications of Natural Language to Information Systems*, 40 – 51, IOS Press

Jaakkola, H. and Thalheim, B. (2003): Visual SQL – High Quality ER Based Query Treatment. *Proceedings of Conceptual Modeling for Novel Application Domains*, LNCS **2814**, 129 – 139, Springer Verlag.

Järvelin, K., Niemi, T., and Salminen, A. (2000): The visual query language CQL for transitive and relational computation. *Data & Knowledge Engineering* **35**, 39 – 51.

Kardovácz, Z.T. (2005), On the Transformation of Sentences with Genetive Relations to SQL Queries. *Proceedings of the 10th International Conference on Applications of Natural Language to Information Systems (NLDB 2005),* LNCS **3531**, 10 – 20, Springer Verlag.

Kapetainos, E., Baer, D., and Groenewoud, P. (2005): Simplifying syntactic and semantic parsing of NL-based queries in adavanced application domains, *Data & Knowledge Engineeing Journal* , **55**, 38 – 58.

Kao, M., Cercone, N., and Luk, W.-S. (1988): Providing quality responses with natural language interfaces: the null value problem. *IEEE Transactions on Software Engineering*, **14** (7), 959 – 984, IEEE Press.

Kate, R.J. and Mooney, R.J. (2006): Using String-Kernels for Learning Semantic Parsers. *COLING/ACL Proceedings*, Sydney, 2006, 913-920.

Kop Ch: (2009a): Visualizing Conceptual Schemas with their Sources and Progress. *International Journal on Advances in Software, International Academy, Research and Industry Association (IARIA)*, **2** (2 u. 3), pp. 245-258.

Kop, Ch. (2009b): Continuous conceptual schema quality checking. *Proceedings of the 4th International Conference on Software and Data Technologies*, 186 – 193, INSTICC Press.

Lindland, O., Sindre, G., and Solvberg, A. (1994): Understanding Quality in Conceptual Modeling, *IEEE Software March 1994*, 29 – 42, IEEE Press.

Mayerthaler, W., Fiedl, G., Winkler, Ch. (1998): *Lexikon der Natürlichskeitstheoretischen Morphosyntax*, Stauffenburg Verlag, Tübingen.

Meng, H.M. and Siu, K-Ch. (2002): Semiautomatic Acquistion of Semantic Structures for Understanding Domain-Specific Natural Language Queries. IEEE Transactions on Knowledge and Data Engineering, **14** (1), 172 – 181, IEEE Press.

Moody, D. (1996): Graphical Entity Relationship Models: Towards a More User Understandable Representation of Data. *Proceedings of the 15th International Conference on Conceptual Modeling, Cottbus, Germany, Lecture Notes in Computer Science (LNCS)*, **1157**, 227 – 245, Springer Verlag.

Moody, D. (2005): Theoretical and practical issues in evaluating quality of conceptual models: current state and future directions. *Data & Knowledge Engineering*, **55**, 243 - 276.

Owei, V., Rhee, H-S., and Navathe, S. (1997): Natural Language Query Filtration in the Conceptual Query Language. *Proceedings of the 30th Hawaii International Conference on System Science*, **3**. 539 – 550, IEEE Press.

Owei, V., Navathe, S. (2001): Enriching the conceptual basis for query formulation through relationship semantics in databases. *Journal of Information Systems* Vol 26, 445 – 475, Elsevier Publ. Company.

Panchenko, O., Müller, St., Plattner, H., Zeier, A., (2011): Querying Source Code Using a Controlled Natural Language. *Proceedings of the Sixth International Conference on Software Engineering Advances (ICSEA)*, pp. 369 – 373.

Penn-Treebank TagSet: http://www.cis.upenn.edu/~treebank/. Accessed 13. Oct. 2011.

Rumbaugh, J., Blaha, M., Premelani, W., Eddy, F., and Lorensen, W. (1991): *Object-Oriented Modeling and Design*. Prentice Hall International Inc. Publ. Company.

Sang, E.F.T.K. and Buchholz, S. (2000) Introduction to the CoNLL-2000 Shared Task: Chunking. *Proceedings of CoNLL-200 and LLL-2000*, 127-132.

Sartori F. and Palmonari M. (2010): Query Expansion for the legal domain – a case study from the JUMAS project. *Proceedings of the 4th International Workshop Ontology, Conceptualization and Epistomology for Information Systems, Software Engineering and Service Sience (ONTOSE 2010),* Lecture Notes in Business Information Processing (LNBIP), **62**, 107 – 122, Springer Verlag.

Stratica, N., Kosseim, L., and Desai, B.C. (2005): Using semantic templates for a natural language interface to the CINDI virtual library. *Data & Knowledge Engineering*, **55**, 4 – 19.

Tang, L.R. and Mooney, R.J. (2001): Using Multiple Clause Constructors in Inductive Logic Programming for Semantic Parsing. *Proceedings of the 12th European Conference on Machine Learning (ECML-2001),* 466-477.

Terwillinger, J.F., Delcambre, L.M., and Logan, J (2007).: Querying through a user interface. *Data & Knowledge Engineering,* **63**, 774 – 794.

Toutanova, K., Klein D., Manning, C.D., and Singer, Y. (2003) : Feature rich part-of speech tagging with a cyclic denpendency network. *Proceedings of HLT-NAACL*, 252 – 259.

Wong, Y.W. and Mooney, R.J. (2006): Learning for Semantic Parsing with Statistical Machine Translation. *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT/NAACL-2006),* New York, 439-446