Query Based Multicontext Based Browsing: a Technical Report

Julien Tane

September 29, 2004

Abstract

Formal Concept Analysis [3] is an algebraic framework suitable for different knowledge processing tasks. It has been widely used in a variety of domains. The current approaches use one lattice (or a nested line diagram) to organize objects according to their properties. However, it is often difficult to change the data representation to adapt it to the goals of the visualisation. It is therefore crucial to ease the interaction with the data for Formal Concept Analysis applications. To tackle this issue, we introduce a new artifact: the query based multicontext. The goal of this report is to address the different issues which play an important role for browsing with the query based multicontext. We first give its definition and describe some of its basic properties. Then, to demonstrate its flexibility, we use it in combination with ontologies to create a knowledge base browsing framework which we illustrate by describing diverse useful interactions with its interface. Furthermore, we discuss our approach by comparing it to the current state of the art in Formal Concept Analysis.

0.1 Introduction

With the success of the Internet, browsers have become a very natural way of searching information in the web. Wordnet gives two sense of the word "browser" :

1. browser – (a viewer who looks around casually without seeking anything in particular)

2. browser, web browser – (a program used to view HTML documents)

If we look at the first meaning of the word, we notice that it does not apply completely to the use Internet user make of a browser. Sometimes, user look for very specific information, but do not know how to find it easily. For this, there are search engines, to which one submits a query to receive a certain number of links to browse from in order to find the information one seeks. This kind of queries are usually based on specific keywords appearing in the text of the web pages found. They do not make use of the semantic information contained in the web pages. In order to remedy to that problem, some search engines are based on content directories, which classifies document about the same topic together. The advantage of a directory is the possibility of browsing through it. As language and data mining technologies evolve, the research made crucial steps to enhance the classification of data using background knowledge. Moreover, the W3C put in later years much effort into the creation of the next Web generation: the Semantic Web as "an extension of the current web in which information is given well-defined meaning" (see[5]). This implies that the creation of browsers capable of exploring and interact with the data using semantic information will become even more crucial.

In order to store and exchange semantic information, diverse kinds of knowledge representation frameworks have been devised (Concept(ual) Graphs [6], Description Logics, RDF-based ontologies [7] ...). All these frameworks contain a conceptual level and an instance level which we will respectively call *ontology* and *fact base*. We will call knowledge base the structure combining the two. The ontology contains the concepts and the relations between them occurring in the domain, whereas the fact base contains statements describing the instances and their relations using a given ontology.

Knowledge bases are usually complex structures, where both conceptual and instance level interplay. In order for users to best understand and use their data, a mean for visualizing the diverse crucial aspects is indispensable. On the conceptual level, a special type of relation plays a major role: the subsumption hierarchy, that is the hierarchy organizing the concepts according to their generality. Moreover, this hierarchy is coupled to an instantiation function which ascribes to each concept the set of instances corresponding to it. Therefore, a browsing tool should be capable of processing and displaying subsumption hierarchies as well as the instantiation function.

If we take a look at the literature, we will see that different applications have been designed and implemented to help browse knowledge bases. However, most ontology browsers (e.g: OntoEdit[1], Protege[2], ...) display hierarchies using trees. While trees have the advantage of being somewhat compactly displayable, they are not very well

[1] http://www.ontoprise.de/products/ontoedit
[2] http://protege.stanford.edu/

suited for data where objects can easily belong to different domains or when concepts may have multiple superconcepts. Some others (see for example, the OIModeller in the KAON framework) use a graph based representation, which looses then the hierarchical representation.

Since its beginning, in the 80's, Formal Concept Analysis has been applied in different fields such as Knowledge Representation or Data Mining. In the domain of the visualization of ontologies, concept lattices offer an elegant solution to some of the drawbacks of trees or graph based visualization techniques. With Formal Concept Analysis, it is possible to display multiple inheritance and multiple instantiation in a natural way. For examples of this, we refer to the following papers on scaling: [4].

Knowledge bases do not only consist of the aforementioned relations: subsumption hierarchies and instantiation functions. A number of non hierarchical relations appear in knowledge bases, for example a given fact base could code the fact that a given person works with another one. Since this kind of relations is not hierarchical in its nature, a way has to be found as how to cope with these. Since the essence of Formal Concept Analysis is to represent binary relations through lattices. Here, again it proves an interesting way of visualizing non hierarchical relations.

However, up to now, most traditional techniques display only one lattice or sets of lattices but do not allow the interaction with the objects or attributes in order to modify the visualisation according to the needs of the situation. To remedy to this issue, we will present, in this paper, an approach based on the combination of Formal Concept Analysis and ontologies which allowed us to create an ontology/FCA-based browsing application to manage web resources for an E-Learning domain, see [8].

The basis of our approach is the use of a new artifact to represent complex data using queries. As we will see in the follwing section, this artifact, called *query based multicontext*, is composed of a family of contexts, whose index set is composed of tuples of three queries. For a given triple, by evaluating the queries, we obtain a new context. We will then show how this new artifact allows us to formalize the navigation between the diverse lattices of interest. Informally, a context of our multicontext corresponds to the current lattice displayed by the application. Given interactions with the interface, which we describe later in this paper, then allow the definition of a new query triple, and therefore a new context.

In the first section, we describe our multicontext by defining it formally. We present basic functionalities and constructors, which are useful for the definition in the following section. To support our claims, we describe formally the way the data in knowledge base can be displayed by combining the query based multicontext with ontologies. The next section addresses the issue on how to navigate through a knowledge base using the graphical user interface. It studies different kinds of user interaction which allow the changing from a lattice to another one by defining a new tuple of three queries. We will illustrate this through a step by step example of how the interface can be used to browse the knowledge base.

0.2 The query based multicontext

In this section, we will discuss the query based multicontext, an artifact that we define in this section and which is central to this article. We will start by recalling a first multicontext structure presented in [9]. We reprint here its definition.

Definition 1 *[9] A* **multicontext of signature** $\sigma : P \rightarrow I^2$*, where I and P are non-empty sets, is defined as a pair* (S_I, R_P) *consisting of a family* $\mathcal{S}_I := (S_i)_{i \in I}$ *of sets and a family* $\mathcal{R}_P := (R_p)_{p \in P}$ *of binary relations with* $R_P \subseteq S_i \times S_j$ *if* $\sigma p = (i, j)$*.*

There are many possible uses of such a structure, it represents a family of contexts, indexed by the set P, which specifies through σ which sets of objects and attributes will be used. However, this kind of multicontext is still too general for our purpose. It does not specify a structure on the diverse aspects defining a given context. We need to describe more precisely the sets of elements chosen as objects or attribute sets. We were interested in displaying the concept lattice representing a context with a complex definition. We wanted to visualise the context composed of the researchers of given institutions: "AIFB", "LS3" and "FZI" as object set, with certain topics of interests: "Data Mining", "NLP" and "Knowledge Representation" as attribute set and with the relation defined as the researchers which have a publication on one of the given topic. This implied defining a complex query on a knowledge base. We were also interested in how one could define such a context by interacting with some other context representation of the knowledge base. This led us quite naturally to the definition of a query based multicontext, which we will define in the following paragraph. For this, we adapted the preceding definition in order to give a maximum control on how a given formal context should look like, while preserving index sets which allow the manipulating of the multicontext on its intensional level. Indeed as we will shortly see, using queries to define the object set, attribute set and incidence relation, creates a new structure with interesting properties. But let us first define the query based multicontext.

0.2.1 Query Based Multicontext Definition

Let us first now expose the formal definition of our query based multicontext using the notations of [3] which we will use until the end of this article.

Definition 2 *Let* Ω *be a set, called universe, and* L_1 *and* L_2 *two query languages, whose queries, i.e their elements, return sets of elements of* Ω*, respectively sets of pairs of elements of* Ω*. Let* $\mathbb{P} := L_1 \times L_1 \times L_2$*. We call an element p of* $\mathbb{P}$*, a context index. Moreover we define two evaluations functions named "eval" on* L_1 *and respectively* L_2*, which return for a given query in* L_1*, respectively* L_2*, the result of the evaluation of the query.*

For a context index $p = (q_1, q_2, q_3) \in \mathbb{P}$*, we can define its induced query-based context as:* $\mathbb{K}_p := (eval(q_1), eval(q_2), eval(q_3) \cap (eval(q_1), eval(q_2)))$*. Now let us define a query based multicontext with index set* $\mathbb{P}$ *as the set:* $\{\mathbb{K}_p | p \in \mathbb{P}\}$*.*

We can now see that it is very easy to map our query based multicontext definition on the multicontext of definition 1. This is simply done by defining P:=$\mathbb{P}$, and I := L_1. Then $\forall p = (q_1, q_2, q_3) \in P$, we set $\sigma(p) = (q_1, q_2)$ and $\mathcal{R}_p := eval(q_3) \cap (eval(q_1) \times eval(q_2))$. The set S_i indexed by the elements of I are the results of the evaluation of q_1 and q_2, or put more formally $\forall q_1 \in L_1, \quad S_{q_1} := eval(q_1)$. In other words, our query based multicontext is a specific case of the multicontext defined in [9]. However, the supplementary information that we have in the query based multicontext allows us to manipulate its context more easily. Indeed, it is possible to define certain operators on the context index, as well as simple generic constructor. Depending on the query languages used for L_1 and L_2, it is also possible to study the relations between contexts of our multicontext through the use of there context index. This opens a wide range of interesting questions as to which query language suits which application.

0.2.2 Some Generic Context Construction Methods

We will now introduce some generic context index constructors which will be used in later parts of this report. They correspond to typical configurations such as creating the context of a hierarchy or of a Join. Their purpose is to allow the easy construction of a context index from given queries, but they also allow a more easily understandable representation of a given context index. Therefore, we define the notations for the created context index. Moreover, it is also interesting and important to see how these constructors interact with each other.

In [3], diverse context constructions operators were defined, such as dual, apposition and subposition[3]. We want to be able to use equivalent operators on the context index, in order to have a kind of calculus on the elements of $\mathbb{P}$. Of course the dual of a context index is trivial. But if we want to define apposition and subposition, we need to add a disjunct union operator as in [3]. But, introducing the disjunction operator creates a problem with our definition where we use a single entity set Ω. We have not yet found a clever theoretical solution to this problem, however, we solve it pragmatically and programmatically by keeping track of the place and query from which the entity has been created[4]. For two sets A and B, we use the notation $disjoint(A, B) := \dot{A} \cup \dot{b}$.

In table 1, we define some useful generic query primitives, which we consider from now on contained in the query languages L_1 and L_2. In order to simplify the table, we will use the following conventions: $q_1, q_2 \in L_1$ and $q_3, q_6 \in L_2$ and R is a binary relation over Ω, i.e $R \subseteq \Omega \times \Omega$.

In [3], diverse context constructions were defined, such as We can now define the operations from [3]: dual, apposition and subposition.

Definition 3 *For $p_1 := (q_1, q_2, q_3)$ and $p_2 := (q_4, q_5, q_6)$ in $\mathbb{P}$, we define the following operators:*

- *dual:* $p_1^d := (q_2, q_1, q_3^{-1})$
- *apposition: if $q_1 = q_4$ then $p_1|p_2$:=*$(q_1, \dot{q_2} \cup_q \dot{q_5}, \dot{q_3} \cup_q \dot{q_6})$.

[3]We will not consider the complementation operation in this paper.

[4]This is done in exactly the same way in [3]

Notation	evaluation
L_{1gen}	Result in Ω
$q_1 \cap_q q_2$	$eval(q_1) \cap eval(q_2)$
$q_1 \cup_q q_2$	$disjoint(eval(q_1), eval(q_2))$
$dom(q_3)$	$\{a \in \Omega \mid (a,b) \in eval(q_3)\}$
$range(q_3)$	$\{b \in \Omega \mid (a,b) \in eval(q_3)\}$
$q_3(q_1)$	$\{b \in \Omega \mid \exists a \in eval(q_1), (a,b) \in eval(q_3)\}$
$q_3^{-1}(q_1)$	$\{a \in \Omega \mid \exists b \in eval(q_1), (a,b) \in eval(q_3)\}$
$R^*(q_1)$	$\{y \in \Omega \mid \exists x \in eval(q_1)(x,y) \in eval(R^*)\}$
L_{2gen}	Result in $\Omega \times \Omega$
$graph(R)$	$\{(a,b) \in \Omega \times \Omega \mid (a,b) \in R\}$
R^*	$\{(x,y) \in \Omega \times \Omega \mid \exists x_1, ..., x_n \in \Omega, xRx_1R \ldots Rx_nRy\}$
$q_3 \cup_q q_6$	$disjoint(eval(q_3), eval(q_6))$
$q_3 \cap_q q_6$	$eval(q_6) \cap eval(q_6)$
q_3^{-1}	$\{(b,a) \in \Omega \times \Omega \mid (a,b) \in eval(q_3)\}$
$q_3 \bowtie_{q_1} q_6$	$\{(a,c) \in \Omega \times \Omega \mid \exists b \in eval(q_1), ((a,b) \in eval(q_3) \wedge (b,c) \in eval(q_6))\}$

Table 1: The table defining the function symbols for the generic query language L_{1gen} and L_{2gen}

- *subposition: if* $q_2 = q_5$ *then* $\frac{p_1}{p_2}$*:=*$(\dot{q_1} \cup_q \dot{q_4}, q_2, \dot{q_3} \cup_q \dot{q_6})$.

With these supplementary context index operators, we are able to define some generic constructors that we use later in this article to create our browsing framework. In some cases, it is also useful to consider that the results of some queries are included in a subset of Ω. Hence, the following definition:

Definition 4 *Let A be a subset of* Ω*, i.e* $A \subset \Omega$*, we call a query* $q \in L_1$*, a query over the set A, if* $eval(q_1) \subset A$.

The purpose of this will be made clear from the following sections.

Hierarchies

The first interesting constructor which should be characterised in our framework, are hierarchies. Of course, the coding of hierarchies in Formal Concept Analysis is quite natural.

Definition 5 *For a query* $q_{\mathfrak{H}}$ *over a partial ordered set* $(\mathfrak{H}, \leq)$*, we define the hierarchy based context index* $Order(q_{\mathfrak{H}}, \leq)$ *for* $(q_{\mathfrak{H}}, \leq)$ *as:*
$Order(q_{\mathfrak{H}}, \leq) := (q_{\mathfrak{H}}, q_{\mathfrak{H}}, graph(\leq))$.

Instantiation Hierarchies

The constructor we will consider here requires an instantiation function, to which we give a very broad definition.

Definition 6 *For two subsets C and I of Ω [5] a function ι defined on C and with values in $\mathfrak{P}(I)$ is called an instantiation function. (C, I,ι)) is then called an* instantiation triple.

Let us now we consider the context of instantiation hierarchies. The instantiation function can take many forms, not just the natural subsumption hierarchy. For example, the relation between the topics and publications can be seen as an instantiation function: a given topic maps to a set of publication on this topic. Therefore, the term concept and instance here are more general than the one used in the knowledge base definition.

We can now define two kinds of instantiation hierarchies depending on whether the query is defined on the concept set or on the instance set:

Definition 7 *For an instantiation triple (C, I,ι)) and for q_C a query over the set C, we define the* instance hierarchy context index for concepts from $q_C$as:
$InstH(q_C, \iota) := (\iota(q_C), q_C, graph(\iota)^{-1})$.

For an instantiation triple (C, I,ι)) and for q_I a query over the set I, we define the *concept hierarchy context index for the instances from q_I*as:
$ConcH(q_I, \iota) := (q_I, \iota^{-1}(q_I), graph(\iota)^{-1})$.

Subsumption/Instantiation Hierarchies

As we said earlier instantiation relation is usually coupled with a subsumption relation on the concept set. But all instantiation triple are not compatible with the hierarchy on the concept set. For this reason, we formalize the compatibility in the following definition:

Definition 8 *Let (C, I,ι) be an instantiation triple. An instantiation triple is compatible with the partial order $(C, \leq)$ if $\forall c, c_1 \in C, c \leq c_1 \Longrightarrow \iota(c) \subset \iota(c_1)$.*

Now we want to define a context, which allows us to combine hierarchies with instantiation functions. Again there are two kinds of context indexes depending on which set is the query actually done.

Definition 9 *Let (C, I,ι) be an instantiation triple compatible with the partial ordered set $(C, \leq)$. For $q_C \in L_1$ be a query on C, and $q_I \in L_1$ a query on I, the* subsumption/Instantiation context index $Sub/Inst(q_C, \leq, \iota)$ for the concepts of q_C is defined as:
$SI(q_C, \leq, \iota) := (q_C \cup_q \iota(q_C), q_C, graph(\leq) \cup_q graph(\iota)^{-1}) = \frac{Order(q_C, \leq)}{InstH(q_C, \iota)}$

and the *subsumption/Instantiation context index $Sub/Inst(q_C, \leq, \iota)$ for the instances of q_I* is defined as:
$SI(q_I, \leq, \iota) := (\iota^{-1}(q_I) \cup_q q_I, \iota^{-1}(q_I), graph(\leq) \cup_q graph(\iota)^{-1}) = \frac{Order(\iota^{-1}(q_I), \leq)}{ConcH(q_I, \iota)}$.

The Join Relation Context

Another important constructor is the Join.

[5] For an instantiation triple, we will call C the *concept set*, and I the *instance set*.

Definition 10 *Let q_A,q_B, q_C be three queries in L_1 and let q_{R_1} and q_{R_2} be queries in L_2. The* Joint context index $Join(q_A, q_{R_1}, q_B, q_{R_2}, q_C)$ between q_A and q_C through q_{R_1} and q_{R_2} over q_C is defined as:
$Join(q_A, q_{R_1}, q_B, q_{R_2}, q_C) := (q_A, q_C, q_{R_1} \bowtie_{q_C} q_{R_2})$

Of course it is possible to generalise the Join Relation Contexts to more than two queries of L_2.

The Relation/Instantiation Context

Another interesting context occurs when one wants to organise some group of objects according to a certain classification. This is the purpose of the *Relation/Instantiation Context*.

Definition 11 *Let C and B be two sets, and (C,B,ι) an instantiation triple, then For $q_A, q_B \in L_1$, $q_R \in L_2$, where $eval(q_B) \subseteq B$, we define the Relation/instantiation context index: $RI(q_A, q_R, q_B, \iota)$ as:*
$RI(q_A, q_R, q_B, \iota) := (q_A, q_B, graph(q_R))|Join(q_A, graph(q_R), q_B, graph(\iota)^{-1}, \iota(q_B))$

Up to now we presented the definition of the query based multicontext, gave some generic operators on the queries and on the context indexes. In the following sections, we present a model for a knowledge base and define a mean of querying it by instantiating L_1 and L_2 for the knowledge base model. By combining the generic constructor exposed above with the ontology-based L_1 and L_2, we will be able to create the knowledge browsing framework presented in the fourth section of this article.

0.3 Query-Based Multicontext for Knowledge Bases

Now we turn to the subject of ontologies and knowledge bases, since we want to show that the query based multicontext is an appropriate representation for applications using ontologies. We first give a formal definition of the knowledge base model that we use in this paper. We then propose the two query languages L_1 and L_2, which are needed to define a query based multicontext on a knowledge base. It will then be followed by an example of knowledge base and give a first example of a context index for the knowledge base query-based multicontext.

The purpose of an ontology is to create a sharable computer-usable representation of a specific domain. For this, the conceptual relation between concepts are coded into an ontology language. Among the scientific community diverse artifacts are called ontologies [7]. In the present work, we present the ontology model that we used and was presented in [2]. Let us now expose the necessary parts:

Definition 12 *An ontology is a tuple $Onto =: (\mathfrak{C}, \leq_C, \mathfrak{R}, \leq_R, \sigma)$ where $(\mathfrak{C}, \leq_C)$ and $(\mathfrak{R}, \leq_R)$ are ordered sets and where σ is a mapping from $\mathfrak{R}$ to non empty words over $\mathcal{C}$.*

To a given ontology, it is possible to give a fact base (slightly different but equivalent definition):

Definition 13 *A* fact base FB for an ontology $Onto$ is a tuple $(\mathfrak{I}_C, \iota_C, \mathfrak{I}_R, \iota_R)$ where $\mathfrak{I}_C$ and $\mathfrak{I}_R$ are called respectively instances and property instances, $\mathfrak{I}_R$ is a set of non empty words on $\mathfrak{I}_C$ (i.e $\mathfrak{I}_R \subseteq \mathcal{P}(\mathfrak{I}_C^+)$)) such that ι_C is a mapping from $\mathcal{C}$ to $\mathcal{P}(\mathfrak{I}_C)$, and ι_R is a mapping from (R) defined as:
$\iota_R(r) \in \Pi_{i=1}^{i=|\sigma(r)|} \iota_C(\pi(i, \sigma(r))$ where $|\sigma(r)|$ is the length of the word $\sigma(r)$ and for all words or tuples, $\pi(i, w)$, returns the i-th letter or component. Moreover the following conditions are valid:
$\forall c, c_2 \in \mathfrak{C},\ c \leq_C c_2 \Rightarrow \iota_C(c) \subseteq \iota_C(c_2)$ and $\forall r, r_2 \in \mathfrak{R},\ r \leq_R r_2 \Rightarrow \iota_R(r) \subseteq \iota_R(r_2)$.

We call a a *knowledge base* a couple $KB = (O, FB)$, where O is an ontology and where FB is a fact base for the ontology O.

0.3.1 Ontology Query Language

As we mentioned earlier knowledge bases contain different kinds of entities and relations. In order to make the most out of its structure, we define the two query languages L_1 and L_2 in such a way, that we can make use of the generic context index constructors defined in section 0.2.2.

Before defining them these two languages, we need to set the basis universe Ω Therefore, for the rest of this article, let $\Omega := \mathfrak{C} \cup \mathfrak{I} \cup \mathfrak{R}$ be the universe. As we said earlier, the evaluation function for queries from L_1 return a set of entities of the base universe, where as for queries from L_2, it returns pairs of entities of this base universe. The function symbols[6] of the query language we present here take as parameters queries.

Let us begin by defining L_1, then L_2.

L_1 - *Querying for Entities* The queries of the first query language define which entities of the knowledge base universe should be retrieved. Since there exists in the knowledge base a natural division between some groups of entities, it makes sense to define three sublanguages L_{1C}, L_{1R} and L_{1I} which return respectively concept, relation and instance sets. Table 2 presents the function symbols for building queries for these languages, each having is own section in the table. To simplify the table, we will use the following notations: $q_C, q_D \in L_{1C}$, $q_R \in L_{1R}$, q_I, q_{I_1}, $q_{I_2} \in L_{1I}$,$i, j \in \mathbb{N}$ with $i \neq j$ and finally $c_1, \ldots, c_n \in \mathfrak{C}$, $i_1, \ldots, i_n \in \mathfrak{I}$, $r_1, \ldots, r_n \in \mathfrak{R}$. All three languages are built inductively from the ground terms defined in the second row of its section. Note that these function symbol may make use some query belonging to one of the two other languages. For example, $\exists_C q_R(j : q_C)$ is built using two queries: q_R and q_C belonging to L_{1R} and L_{1C} respectively.

Finally, the query language $L1$ is the language the language recursively defined by queries of L_{1C}, L_{1R}, L_{1I} or of queries built from these query set as term for the operation symbols defined in table 1.

L_2 - *Querying for Entity pairs* In a similar way, table 3 presents some new function symbols for a query language which returns pairs of elements. As for L_1, we consider L_2 as the language created through the use of these symbols in addition to the function symbols defined in table in table 1.

[6]There are three exceptions: $\{c_1, \ldots\}$, $\{i_1, \ldots\}$ and $\{r_1, \ldots\}$.

Notation	Meaning	Query Result
L_{1C}	returns (concept names)	Results: Concept Sets
ALLCONCEPTS	returns all the concepts	$\mathfrak{C}$
$\{c_1, \ldots, c_n\}$	the set of concepts composed of $\{c_1, \ldots, c_n\}$	$\{c_1, \ldots, c_n\}$
$\leq_C q_C$	all subconcepts of elements of q_C	$\{d \in \mathfrak{C} \vert \exists c \in eval(q_C), d \leq_C c\}$
$\geq_C q_C$	all superconcepts of elements of q_C	$\{d \in \mathfrak{C} \vert \exists c \in eval(q_C), d \geq_C c\}$
$q_C \sqcap_C q_D$	intersection of the sets of elements of q_C and q_D	$\{e \in \mathfrak{C} \vert \ e \in eval(q_C) \wedge \ e \in eval(q_D)\}$
$q_C \sqcup_C q_D$	union of the sets of elements of q_C and q_D	$\{e \in \mathfrak{C} \vert e \in eval(q_C) \vee e \in eval(q_D)\}$
$concepts(q_I)$	the set concepts of which the instances in q_I are instances	$\{e \in \mathfrak{C} \vert \exists i \in eval(q_I) \iota(e)\}$
$\exists_C.q_C$	concepts attached to some concept of q_C through some relation	$\{e \in \mathfrak{C} \vert \exists m, n \in \mathbb{N}, m \neq n, \exists c \in q_C \exists r \in \mathfrak{R}, e \in \pi(n, \sigma(r)) \wedge c \in \pi(m, \sigma(r))\}$
$\exists_{Ci} q_R(j : q_C)$	concepts at a position i in one of relation of q_R related to a concept of q_C at position j	$\{d \in \mathfrak{C} \vert \exists c \in eval(q_C), \ \exists r \in eval(q_R), c = \pi(j, \sigma(r)) \wedge d = \pi(i, \sigma(r))\}$
L_{1R}	returns (relation names):	Results: Relation Sets
ALLRELATIONS	all the relations	$\mathfrak{R}$
$\{r_1, \ldots, r_n\}$	the set of relations composed of $\{r_1, \ldots, r_n\}$	$\{r_1, \ldots, r_n\}$
$\leq_R q_R$	all subrelations of elements of q_R	$\{s \in \mathfrak{R} \vert \exists r \in eval(q_R), s \leq_C r\}$
$\geq_R q_R$	all superrelations of elements of q_R	$\{s \in \mathfrak{R} \vert \exists r \in eval(q_R), s \geq_C r\}$
$\exists_R.q_C$	relations attached to a concept of q_C	$\{r \in \mathfrak{R} \vert \exists c \in eval(q_C), \exists n \in \mathbb{N}, c = \pi(n, \sigma(r))\}$
$\exists_R q_R(j : q_C)$	the set of relations where there exists concept belonging to c compatible with its signature	$\{r \in \mathfrak{R} \vert \exists c \in eval(q_C), r \in eval(q_R), c = \pi(j, \sigma(r))\}$
$\exists_R.(i : q_{I_1}, j : q_{I_2})$	relations where some instance of q_{I_1}, q_{I_2} appear at position i and j respectively	$\{r \in \mathfrak{R} \vert \exists i_1 \in eval(q_{I_1}), \exists i_2 \in eval(q_{I_2}) r \in eval(q_R), \exists w \in \iota_R(r), i_1 = \pi(i, w) \wedge i_1 = \pi(j, w)\}$
L_{1I}	returns (instance names):	Results: Instance Sets
ALLInstances	returns all the instances	$\mathfrak{I}_C$
$\{i_1, ..., i_n\}$	the set of instances $\{i_1, ..., i_n\}$	$\{i_1, ..., i_n\}$
$Inst(q_C)$	the set of instances of concepts belonging to q_C	$\{i \in \mathfrak{I}_C \vert i \in \iota(eval(q_C))\}$
$q_C \sqcap_I q_D$	intersection of the sets of instances of q_C and q_D	$\{i \in \mathfrak{I}_C \vert i \in \iota_C(eval(q_C) \wedge i \in \iota_C(eval(q_D))\}$
$q_C \sqcup_I q_D$	union of the sets of instances of q_C and q_D	$\{i \in \mathfrak{I}_C \vert i \in \iota_C(eval(q_C)) \vee i \in \iota_C(eval(q_D))\}$
$\exists_I q_R(j : q_C)$	set of instances related to some instance of one concept of q_C through some relation of q_R	$\{d \in \mathfrak{I}_C \vert \exists c \in eval(q_C), \exists r \in eval(q_R), c \in \sigma(r)\}$
$\exists_I q_R(j : q_I)$	set of instances related to some instance in q_I through some relation out of q_R	$\{d \in \mathfrak{I}_C \vert \exists i_1 \in eval(q_I), \exists r \in eval(q_R), (d, i_1) \in \iota_R(r)\}$

Table 2: The table defining the function symbols of query languages L_{1C}, L_{1R} and L_{1I}.

Notation	Meaning	Query Result
$Inst_{ij}(q_R)$	the sets of pair of instances for which one relation of q_R holds	$\{(i_1, i_2) \in \mathfrak{I}_C \times \mathfrak{I}_C \vert \exists r \in eval(q_R), (i_1, i_2) \in \iota_R(r)\}$
INSTANTIATION	the instantiation function for concepts	$graph(\iota_C) := \{(c, i_1) \in \mathfrak{C} \times \mathfrak{I}_\mathfrak{C} \vert i_1 \in \iota_C(c)\}$
$\leq_C$	the graph of $\leq_C$	graph($\leq_C$)
$\leq_R$	the graph of $\leq_R$	graph($\leq_R$)

Table 3: The table defining the knowledge base specific function symbols for the query language L_2

A Note on Complexity

For our browsing application, the efficiency of the computation of the queries played a great role. For that reason, we chose query languages, whose queries are computable in polynomial time. It can be proven that all these queries are polynomial. The reader interested in this topic is referred to [1] which is a good reference for the complexity of query languages. The query languages[7] presented in this article belong to the family of the *positive existential queries, which is equivalent to the family of* conjunctive queries with union. It should also be noted that the given queries are also language are monotonic (see [1]).

In 0.5.3, we also provide a pseudo code implementation of the proposed query languages. This somewhat naive implementation will be replaced by the use of the query language. We are still looking at the possibilities to insert negation. But we have not fully addressed that issue yet.

However, it should be noted that the query languages are declarative. They do not suppose a special implementation and the three queries of a context index are not supposed to be implemented independently from each other.

0.3.2 An Example Ontology

In order to illustrate our approach, we will now present an example of knowledge base and of one context index defined using the query languages presented in the previous paragraph. The example knowledge base we use is describes universities. The following table shows part of the ontology's concept set $\mathfrak{C}$ and relations $\mathfrak{R}$.

$\mathfrak{C}$	Person, Student, Professor, Teaching Staff, Tutor, Assistant, PhDStudent, Topic, Publication, Institution, Country, Region, Conference, Date, Event...
$\mathfrak{R}$	takesPartIn, hasWritten, holdsLecture, hasTeachingRole, belongsTo, isPartOf, publishedAt, takesPlaceAt, followsLecture, ...

The fact base for this knowledge base contains, for example, the persons work-

[7] This might not be true for queries defined with the help of the transitive closure operator.

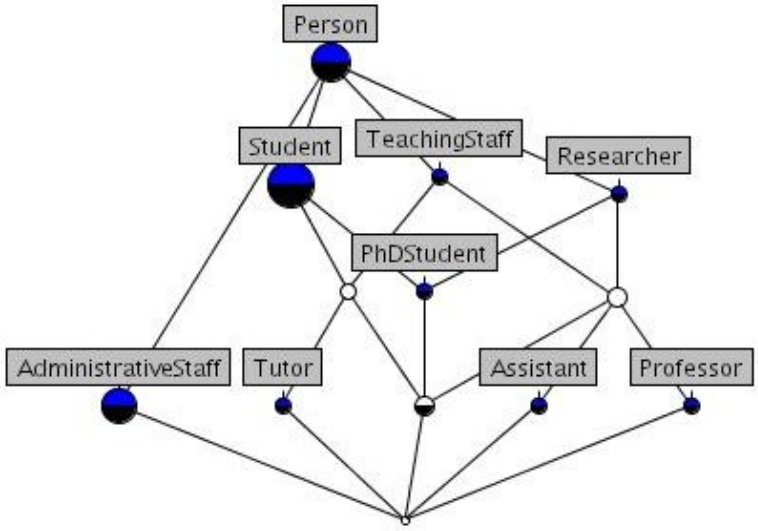

Figure 1: The subsumption-instantiation lattice for all the subconcepts of "Person": $SI(\leq_C \{Person\}, \leq_C, INSTANTIATION)$.

ing at the university, their roles, their publications, their lectures... For example, you can find statements like: Professor("Gerd Stumme"), Lecture("Knowledge Discovery Lecture"), holdsLecture("Gerd Stumme","Knowledge Discovery Lecture"). Meaning that"Gerd Stumme" is an instance of "Professor", "Knowledge Discovery Lecture" is a "Lecture", "Gerd Stumme" holdsLecture "Knowledge Discovery Lecture". The first two statements were instantiations whereas the third one is a relation-instance.

Now,using the query languages defined for knowledge bases, it is easy to write the context index of the subsumption-instance hierarchy of the subconcepts of "Person": $SI(\leq_C \{Person\}, \leq_C, INSTANTIATION)$. The diagram in Figure 1 displays the concept lattice corresponding to this context index. In the following section, we address the issue of defining a knowledge base browsing framework for such a knowledge base.

0.4 An Ontology-based Formal Concept Analysis Browsing Framework

Among the applications of the query based multicontext, we will now describe the creation of a browsing framework for Formal Concept Analysis. This browsing frame-

work is based on the combination of the query based multicontext and knowledge bases presented in the previous section. We will describe our approach by drawing a parallel with typical web browsers such as Firefox or Opera[8]. A web browser displays one page at a time, by fetching it from a location specified by its URL (Unified Resource Locator). In some cases, there is no actual page stored on the server, but is automatically generated by the server. In our approach, we feed our browser with a context index $p \in \mathbb{P}$ instead of an URL, this causes the query based multicontext based browser to display the Hasse diagram of the concept lattice ($\underline{\mathfrak{B}}(\mathbb{K}_p)$) of the given context index. In a web browser, a user can change the displayed page by simply entering a new URL or interacting with the page to get some relevant new URL. The browser the displays the new URL. In our browsing framework, both entering and interacting are also possible. The user can enter a new context index and the browser displays its lattice. But the user can also interact with the displayed lattice in order to create a new context index and again the browser displays its lattice. In order to describe the way our browser works, we describe the kind of interaction with the lattice which result in the creation of a new context index. This description is done by first presenting the seletion mechanism, that is what and how the objects are selected from the lattice. Then we introduce the workflow mechanism, which allows the creation of different context index depending on the selected objects and the browsing mode.

0.4.1 Selection

As we mentioned earlier, we want to characterise the selection mechanism, in other words what can be selected and how it can be selected. In the context of a Formal Concept Analysis browsing application, there are many types of objects which can be selected but they can be separated in two main classes: lattice elements on the one hand and entities of the entity universe Ω on the other. Before we address the question of what is to be selected, we want to specify how we can make a selection. We will consider only one click on a lattice element[9]. But we will consider two kinds of clicks: "left-click" and "right-click". The former corresponds to a selection following a given mode, whereas the latter corresponds to the selection of the given set, but shows a context menu with diverse actions on this set. In order to understand our approach, we first expose the diverse modes, then we describe the result of a "left-click" on an element of the lattice, then of a "right-click".

Click Mode In order to formalize the interaction with the lattice, we define the notion of click mode. A click mode is the type of lattice element which is selected from the lattice. When clicking on a lattice element, different types of objects might be interesting to select: concepts, concept intents, concept extents, extent contingents, intent contingents, labels, filters, ideals, concepts sets... Table 4 contains a list of lattice elements which can be selected by using one click. Because we defined the query based multicontext for knowledge bases exclusively on sets, we only consider entity sets. The selection creates a query on the selected set: $\{< elementsoftheselectedset >\}$.

[8] See http://www.free-definition.com/Web-browser.html

[9] We will not address the selection of more than one lattice element.

Table 4: The different click modes and the type of sets you get when clicking

Lattice object	Type selected
label	the singleton set $\{entity\}$
extent	entity set
intent	entity set
extent contingent	entity set
intent contingent	entity set
concept	pair of entity sets
filter	concept set
ideal	concept set
filter $\cup$ ideal	concept set
neighbourhood	concept set

Hence, we consider that for a given selection, the selection mode is chosen. This plays an important role, both when "right-clicks" or "left-clicks" are used. [10].

Left-Click In our browsing interface left-clicking on a node corresponds to the direct selection of the entities as predefined by the chosen mode: concept, intent....

Let us now consider the context induced by the context index: $SI(\leq_C \{Person\}, \leq_C)$ (see Figure 1). In the concept selection mode, left-clicking on the node of the formal concept μ"$PhDStudent''$ means selecting the concept with extent: $eval(Inst(\{\text{"}Ph\dot{D}Student''\}) \cup_q \leq_C \{\text{"}Ph\dot{D}Student''\})$ and intent: $eval(\geq_C \{\text{"}PhDStudent''\} \cap_C \leq_C \{\text{"}Person''\})$, whereas in the extent contingent mode, the same click means selecting the set eval($\{\text{"}PhDStudent''\}$) which of course returns: $\{\text{"}PhDStudent''\}$. Therefore, it is up to the user to decide what he wants to select by choosing the right mode. Again you will find the selection modes in table 4, which work with a simple "left-button" click of a concept or of a label.

Right-Click In the query based multicontext definition, Ω can be anything, but we will consider only the case where underlying universe is a knowledge base. We saw in the previous section that it was possible to query to get to new sets of entities or new relations. We presented two query languages, L_1 and L_2 which delivered queries over concepts, relations or instances of the knowledge base.

These entities allow the direct access of new entity set through queries on other entity sets, like Inst on a concept entity. Therefore, "right-clicking" opens a context menu for more complex actions such as:

1. selecting the set defined by one of the lattice element given above (concept, intent...)
2. selecting one or more entities from these same sets...
3. selecting entities or an entities' set directly accessible from one of these entities

[10]Two possibilities for setting/choosing mode would possible: either external or defined on specific part of the chosen concept (top part of the node, bottom part...)

4. some other action

(1) is equivalent to selecting another mode. (2) corresponds to selecting a subset of the selected set. (3) means: use a query primitive on the given set and select its result, for example on a set concept set, "select InstancesOf", (4) do some action such as displaying a specific context index by using this set.

Let us consider the example lattice of figure 1. Let us first give an example for (2). In the concept extent mode, right-clicking on the node labeled "Researcher" (i.e representing the concept $\mu''Researcher''$) allows for choosing some instances among its extent, for example "Julien Tane", "Professor Stumme", the "some" allows the user determining exactly the instances of interest. An example for (3), right-clicking on Professor and choose "select Instances for Relation¿holdsLecture", and select all Instances. For an example of (4), suppose the mode is: "select intent contingent", right-clicking on the node for formal concept $\mu"Researcher''$ opens a context menu adapted to the set: $\{"Researcher''\}$. Some of the possible actions in the context menu are actions for creating a new context index for the given set (here $\{"Researcher''\}$). For example, the action "display subconcept hierarchy for", would create the context index: Order($\leq_C$ $\{"Researcher''\}$, $\leq_C$), in other word the subconcept hierarchy for the elements of the concept query $\{"Researcher''\}$.

To conclude this section, we have shown diverse example of selection. A selection can be directly used as in (4). But, it can also be used in more complex workflows. In the following paragraph we will expose one of these complex workflows.

0.4.2 Usage Workflow

We just saw how some entity sets can be selected. We will now introduce diverse workflows to create a new context index. In section 0.2.2, we presented different generic constructor for some interesting context index. In our approach, we use these generic constructors as a kind of workflow guideline to create the new context index. Each of these constructors constitutes a browsing mode. The workflows of a given mode can be simple: it involves only a "left-click", or it can be complex, it involves a sequence of selections.

The user can choose one browsing mode among which are: simple relation lattice, Order Mode Subsumption Instantiation lattice, Join Lattice, Relation-Instantiation lattice. To each of these browsing mode is associated a browser workflow. In a given mode, some variables should be replaced by queries. Let us describe the modes:

- the Simple Relation Lattice mode: Three queries (q_1, q_2, q_3) of the new context index are defined one after the other. In other words, the workflow consists in the definition of each of these queries, the new context index is then: (q_1, q_2, q_3).
- the Order Lattice: Only one selection q is needed, but should be of one of the following type: L_1C or L_1R. The returned context index is: $Order(q, \leq_C)$ if $q \in L_1C$ and $Order(q, \leq_R)$ if $q \in L_1R$.

- the Subsumption Instantiation lattice mode: Only one selection is needed (but should belong to L_1C or L_1I). The context index of the new lattice to be displayed is: $SI(q_C, \leq_C, \iota_C)$.

- Join Lattice: here five selection are needed:

  1. an instance query q_A
  2. a relation query q_{R1}
  3. an instance query q_B
  4. a relation query q_{R2}
  5. an instance query q_C

  The returned context index is then $Join(q_A, q_{R1}, q_B, q_{R1}, q_C)$.

- Relation-Instantiation: here four selections are needed:

  1. an instance query q_A
  2. a relation query q_R
  3. an instance query q_B
  4. the instantiation relation q_R

In the following example, we will use the latter one since it is the most complex. But, it is important to note, that it is always possible to select the intent, extent... of a given node of any lattice and display some very simple lattices for it: for example, if the set I chosen builds an instance query q_I (that is the query evaluated to get this set was in L_1I), then it is possible to build the context index $ConcH(q_I, \leq_C)$ by choosing the menu in the context menu.

The goal is to display the concept lattice of the context having persons working at certain institutions as objects, and the publications and their topics as attributes. The incidence relation is then composed of the relation instances of "hasWritten", as well as of the join of "hasWritten" with "isOnTopic". Put formally, this means define the context index:
$RI(q_1, \{hasWritten\}, \{Publication\}, \{isOnTopic\}, \{Topic\})$ where:
$q_1 := \exists_{I1}\{worksAt\}(2 \in \{\text{“}AIFB''\text{, “}FZI''\})$

Diverse steps are needed:

- define object set (shown $InstH(ALLCONCEPTS, \leq_C)$:
  - do a right click[1] on "Person", and choosing in the context menu: "select For Instances for Query with constraint $>$ worksAt(Institution)".
  - do a right click on "Institution" and choose in the list of instances "AIFB" and "FZI".

- choose relation (shown $Order(\exists_R.(\{\text{“}Person''\})\}, \leq_R)$) (see Figure 2):

[7] It opens the context menu.
[8] It opens the context menu.

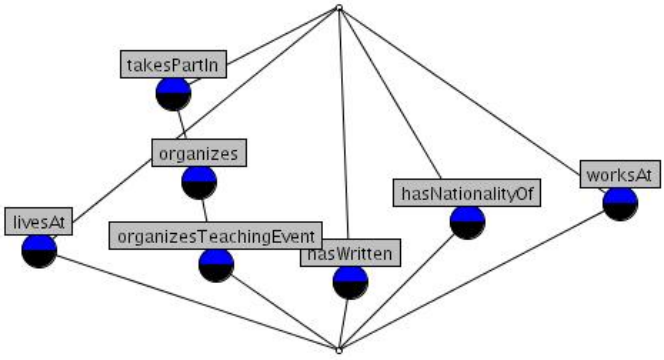

Figure 2: The subsumption lattice of the relations of "Person":$Order(\leq_R \{\exists_R.(\{\text{“}Person''\}, \leq_R)$.

  - do a left click[2] on "hasWritten("Publication")".

- refine "Publication" (shown $Order(\text{“}PUBLICATION'', \leq_C)$):
  - do a right click[1] on "ScientificArticle", and choosing in the context menu: "select For Instances for Query with constraint $>$ "publishedInProceedingOf("Event")

- choose instantiation relation (shown $Order(\exists_R.(\text{“}Publication''), \leq_R)$):
  - do a right click on "isOnTopic(Topic)"

- display $RI(q_1, \{hasWritten\}, \{Publication\}, \{isOnTopic\})$ (see Figure 3)

Some other aspects which play a crucial role for the acceptation of such a browsing interface such as the choice of the lattice display algorithm, of the nodes' and labels' appearance, etc... have not yet been treated here. We plan to extend this report with section concerning these topics.

0.5 Implementation

In this section, we will shortly describe our the basic elements of our ongoing implementation of this browsing framework. Our browsing application mainly sets on two existing Java applications that we extended in order to combine them: the Concept

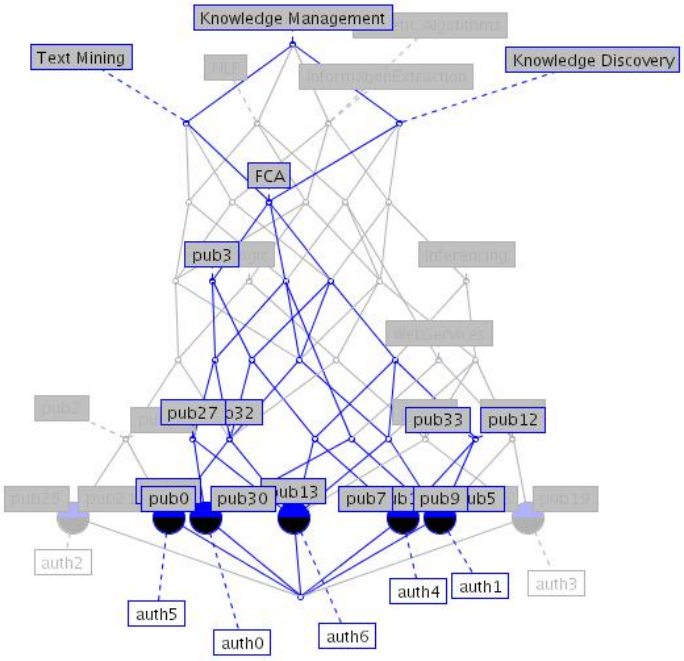

Figure 3: The final lattice of the context index: $RI(q_1, \{hasWritten\}, \{Publication\}, \{isOnTopic\}, \{Topic\})$.

Explorer[11] from Serguey Yevtushenko and the KAON Ontology framework. We will shortly describe them before discussing their combination.

0.5.1 Concept Explorer

The Concept Explorer is a FCA application designed to implement and test diverse algorithms and methods. While it is is a standalone Formal Concept Analysis application, we use it mainly as a Formal Concept Analysis library for data model, algorithms and graphical display. The implementation being quite efficient, it satisfies the need of a FCA browsing application. We made minimum changes to the Concept Explorer in the hope of integrating our ideas in Concept Explorer.

[11] See http://www.sourceforge.net/projects/conexp.

0.5.2 KAON Ontology Framework

The KAON framework can be seen as a graphical user framework with diverse ontology applications all based on a powerful ontology API: the KAON API. It has different kind of applications, such as a graph-based ontology editor: OI-modeler, or a set of ontology learning tools called texttoonto[12]. On top of the KAON framework another tool suite: the Courseware Watchdog has been designed, whose goal is to find, organize learning material. It mainly integrates different kind of tools such as a focused crawler, a interface to a peer-to-peer network, a text clustering application, and the ontology learning tool. This gives a good basis to show how formal concept analysis can be used in cooperation with other tools. In [8] we presented the interaction of the Formal Concept Analysis interface with the other components of the Courseware Watchdog.

A few supplementary words should be said about the KAON Ontology API. This API serves as Facade to diverse ontology back-end types. You may have stored your knowldege base in a database or as a file. This allows for further flexibility since the storing of a knowlege base in itself is independent of its use. Moreover, it allows for multilingual representation of the ontology through the use of a conceptual layer.

0.5.3 Combining ConceptExplorer and KAON

In order to combine the two we load the ontology–instance model. At the request of a user, the application creates the corresponding context. The query languages on the knowledge base presented earlier in this paper, have been implemented on top of the KAON API. We are also exploring the possiblity of using the KAON query language for querying directly knowledge base s.

All along this paper, we presented the query based multicontext, but it is crucial to mention that the evaluation of a context index, does not imply the independant evaluation of its queries. To manage context intensionally using their context index, the latter are very useful. However, in many cases, it would be inefficient to calculate the three queries independantly. Therefore, the engine which evaluates the context index may choose to combine the queries to be more efficient, as long as the resulting context would have been the one obtained through the independant evaluation.

First let us define the API:

A supplementary method on the ontology: getTopConcepts

- Ontology
  - getInstances: returns all the instances the ontology
  - getConcepts: returns all the concepts the ontology
  - getRelations: returns all the relations the ontology
  - getRelationInstances: returns all the relation instances the ontology
- Concept:
  - getInstances: returns the instances of this concept

[12] See http://www.sourceforge.net/projects/texttoonto

  - getSubConcept: returns the subconcepts of this concept
  - getSuperconcept: returns the superconcepts of this concept
  - getRelationsForAtPosition(int j): returns the relations where the concept is at position j
  - getRelations: returns all the relations r where this concept is a component of $\sigma(r)$

- Relation:
  - getSubRelations: returns the subrelations of this relation
  - getSuperRelations: returns the superrelations of this relation
  - getRelationConceptTuple: returns the tuple of concepts for which this relation is defined

- Instance:
  - getParentConcepts

In the following, we give either a simple description on how to implement this query, or we give the pseudo-code for more complex queries. Most of the algorithm are quite straight forward. Of course, this present a very simple possible implementation.

- Concepts
  - ALLCONCEPTS: already part of the API
  - $\{c_1, ..., c_n\}$: add them one after the other to a set.
  - $\leq_C q_C$: please find the pseudocode in algorithm 1.

Algorithm 1 Returns all the subconcepts of the concepts resulting from query q_C

```
nottreated = eval (q_C)
res = ∅
while nottreated ≠ ∅ do
   choose an element e from nottreated
   remove e from nottreated
   add e to res
   add e.getSubConcepts() to nottreated
end while
return res
```

  - $\geq_C q_C$: Same thing as algorithm 1, except for the use of the method: getSuperConcepts() instead of getSubconccepts().
  - $q_C \sqcap_C q_D$: simple set intersection of the results of eval (q_C) and eval (q_D)
  - $q_C \sqcup_C q_D$: simple set union of the results of eval (q_C) and eval (q_D)

concepts(q_I):

Algorithm 2 Returns all the concepts which instances resulting from query q_I

```
nottreated = eval (q_I)
res = ∅
while nottreated ≠ ∅ do
   choose an element e from nottreated
   remove e from nottreated
   add e.getParentConcepts() to res
end while
nottreated = res
res = res.copy()
while nottreated ≠ ∅ do
   choose an element e from nottreated
   remove e from nottreated
   add e.getSuperconcepts() to res
end while
return res
```

  - $\exists_C.C$: Same algorithm as algorithm 2 except that getRelations replaces getParentConcept, and get
  - $\exists_{Ci} q_R(j : q_C)$:

- Relations:
  - ALLRELATIONS: already part of the API.
  - $\{r_1, ..., r_n\}$: add them one after the other to a set.
  - $\leq_R q_R$: same kind of algorithm as algorithm 1, replacing getSubConcept by getSubRelation.
  - $\geq_R q_R$: same kind of algorithm as algorithm 1, replacing getSubConcept by getSuperRelation.
  - $\exists_R.q_C$: same kind of algorithm as algorithm 4, replacing getInstances by getRelations.
  - $\exists_R q_R(j : q_C)$: The algorithm uses is algorithm 3 to evaluate the query.
  - $\exists_R.(i : q_{I_1}, j : q_{I_2})$:

- Instances:
  - ALLINSTANCES: already part of the API.
  - $\{i_1, ..., i_n\}$: add them one after the other to a set.
  - Inst(q_C): The algorithm 4 describes how instantiation can be defined.
  - $q_{I1} \sqcap_I q_{I2}$: simple set intersection of the results of eval (q_{I1}) and eval (q_{I2})
  - $q_{I1} \sqcup_I q_{I2}$: simple set union of the results of eval (q_{I1}) and eval (q_{I2})

Algorithm 3 Returns all the relation belonging to the results of q_R and which are defines as having a concept of q_C at position j

```
allowedrel = eval (q_R)
nottreated = eval (q_C)
res = ∅
for all nottreated ≠ ∅ do
    pick and remove an element e from nottreated
    put e.getRelationsForAtPosition(j) into res
end for
res = intersection (allowedrel,res)
return res
```

Algorithm 4 Returns all the concepts which instances resulting from query q_I

```
nottreated = eval (q_C)
res = ∅
while element in nottreated do
    choose an element e from nottreated
    remove e from nottreated
    add e.getInstances() to res
end while
return res
```

0.6 Conclusion

This technical report presented the query based multicontext and one of the possible application: knowledge based browsing with Formal Concept Analysis. It introduced the definition of the query based multicontext, some of its promissing features and diverse generic context constructors. Then we investigated how this new structure can be used on top of a knowledge base by defining appropriate query languages. Finally, we described a new browsing framework which combines ontologies and Formal Concept Analysis for knowledge base browsing. Future work will be oriented towards other kind of data: databases, data mining results, text collections. Another interesting path of research will be how to integrate other Formal Concept Analysis techniques with the query based multicontext as well as further consequences of the query languages' expressivity.

Bibliography

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison Wesley, Reading, Mass. U.S.A, 1996.

[2] E. Bozsak et al. KAON - Towards a large scale Semantic Web. In *Proceedings of the Third International Conference on E-Commerce and Web Technologies (EC-Web)*. Springer Lecture Notes in Computer Science, 2002.

[3] Berhard Ganter and Rudolf Wille. *Formal Concept Analysis – Mathematical Foundations*. Springer Verlag, Berlin – Heidelberg, 1999.

[4] Bernhard Ganter and Rudolf Wille. *Applications of Combinatorics and Graph Theory to the Biological and Social Sciences*, volume 17, chapter Conceptual Scaling. Springer Verlag, 1989.

[5] Tim Berners Lee, James Hendler, and Ora Lassila. The Semantic Web: A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities. 2001.

[6] John F. Sowa. *Knowldege Representation - Logical, Philosphical and Computational Foundations*. Brooks/Cole, Pacific Grove, CA, U.S.A, 2000.

[7] Rudi Studer and Steffen Staab, editors. *Handbook on Ontologies in Infornation Systems*. Springer Verlag, Berlin – Heidelberg, 2003.

[8] Julien Tane, Christoph Schmitz, Gerd Stumme, Steffen Staab, and Rudi Studer. The Courseware Watchdog: an Ontology-based tool for Finding and Organizing Learning Material. In Ingo Wegener, editor, *Fachtagung Mobiles Lernen und Forschen, 6.11.2003, Universität Kassel*, 2003.

[9] Rudolf Wille. Conceptual Structures of Multicontexts. In *Conceptual Structures: Knowledge Representation as Interlingua, Proceedings of the 4th International Conference on Conceptual Structures, ICCS'96*, pages 23–39, Sydney, Australia, 1996. Springer Lecture Notes in Computer Science.