On Ordering and Indexing Metadata for the Semantic Web

Jeffrey Pound[†], Lubomir Stanchev[¶], David Toman[†,‡], and Grant E. Weddell[†]

[†]David R. Cheriton School of Computer Science, University of Waterloo, Canada

[¶] Computer Science Department, Indiana University - Purdue University, U.S.A.

[‡] Faculty of Computer Science, Free University of Bozen-Bolzano, Italy

1 Introduction

RDF underlies a vision of the Semantic Web in which metadata, consisting of a set of subject/property/object triples, can be associated with web resources denoted by *Universal Resource Identifiers* (URIs) [1]. To support reasoning, there has been a progression of further standards for inferring the existence of additional triples. This is accomplished by adding interpretations for particular RDF properties.

In terms of established reasoning technology, the current best practices for these standards are the *description logic* (DL) based fragments of the OWL web ontology language, called OWL Lite and OWL DL [2]. Building on RDF Schema, they enable a collection of triples to encode more general concepts. This metadata can then be modeled as a set of concept descriptions in a description logic.

In previous work, we introduced the notion of an *ordering description*, a language for specifying strict partial orders over the space of possible concept descriptions in a given DL dialect [3]. These ordering descriptions were then used to build *description indices*, tree-based indices over databases consisting of sets of descriptions, with performance guarantees on query evaluation under particular restrictions. In this paper, we extend our work on ordering descriptions and description indices.

The main contributions of this paper are as follows:

1. We extend the definition of ordering descriptions with three new ordering constructors. One providing an endogenous nested indexing capability, a second as a weaker version of our earlier partition ordering, and the last appealing directly to subsumption relationships. We then discuss these adaptations in the context of the functionality they enable for indexing different classes of concept descriptions;
2. We validate the applicability of our ordering language as a basis for indexing concept descriptions with an experimental evaluation using a prototype implementation of description indices.

The remainder of this paper is organized as follows. The following subsection provides the definitions used in our discussion. Section 2 then presents our revised

definition of ordering descriptions, along with an analysis of their properties under various assumptions of the descriptions being indexed. In Section 3 we report on our experimental results, and Section 4 concludes with a summary and discussion.

1.1 Definitions

We begin with a formal definition of $\mathcal{ALCQ}(\mathcal{D})$, the description logic dialect used in this paper. It should be noted however, that this choice is simply to satisfy our illustrative purposes, and our results are applicable to any description logic dialect with a total linearly ordered domain.

Definition 1 (Description Logic $\mathcal{ALCQ}(\mathcal{D})$).
Let $\{C, C_1, \ldots\}$, $\{R, S, \ldots\}$, $\{f, g, \ldots\}$ *and* $\{k, k_1, \ldots\}$ *denote sets of* primitive concept names, roles, concrete features, *and* constants *respectively. A* concept description *is then defined by the grammar:*

$$D, E \quad ::= \quad f < g \mid f < k \mid C \mid D \sqcap E \mid \neg D \mid \exists R.D \mid (\geq\ n\ R\ D).$$

An inclusion dependency *is an expression of the form* $D \sqsubseteq E$. *A* terminology $\mathcal{T}$ *is a finite set of inclusion dependencies.*

An interpretation $\mathcal{I}$ *is a 3-tuple* $\langle \Delta_I, \Delta_C, \cdot^{\mathcal{I}} \rangle$ *where* Δ_I *is an arbitrary abstract domain,* Δ_C *a linearly ordered concrete domain, and* $\cdot^{\mathcal{I}}$ *an interpretation function that maps each concrete feature* f *to a total function* $f^{\mathcal{I}} : \Delta_I \rightarrow \Delta_C$, *each role* R *to a relation* $R^{\mathcal{I}} \subseteq \Delta_I \times \Delta_I$, *each primitive concept* C *to a set* $C^{\mathcal{I}} \subseteq \Delta_I$, *the* $<$ *symbol to the binary relation for the linear order on* Δ_C, *and* k *to a constant in* Δ_C. *The interpretation function is extended to arbitrary concepts in the standard way.*

An interpretation $\mathcal{I}$ *satisfies an inclusion dependency* $D \sqsubseteq E$ *if* $(D)^{\mathcal{I}} \subseteq (E)^{\mathcal{I}}$. $\mathcal{T} \models D \sqsubseteq E$ *if* $(D)^{\mathcal{I}} \subseteq (E)^{\mathcal{I}}$ *for all interpretations* $\mathcal{I}$ *that satisfy all inclusion dependencies in* $\mathcal{T}$.

For the remainder of the paper, we also use standard abbreviations, e.g., $D \sqcup E$ for $\neg(\neg D \sqcap \neg E)$, as well as the derived comparisons $\leq, >, \geq$, and $=$ on the concrete domain.

2 Ordering Descriptions

From our experiences in considering description indices for some specific problem domains, we have found the need for nested indexing. Nested indexing is an ordering of objects in an abstract domain based on their relationships to other objects. For example, one may want to index an entity set based on a role or abstract feature chain relation to another entity with particular properties. Also, it may be beneficial to define orderings based strictly on hierarchical relationships. This prompts the consideration of some extensions to our ordering language, and

the definition of a weaker version of our partition ordering to allow some basic exogenous nested indexing by partitioning on existential role descriptions.

Below we extend our previous definition of an ordering description, and also relax the definition of the partition ordering as seen in [3]. We then reproduce the definition of a description tree and description index for reference during the discussion. We begin with a comment on notation for obtaining a copy of a concept description with unique primitive concepts, features, and role names. This aids in the definition of the ordering semantics.

Notation 1 We write D^* to denote a description obtained from D by replacing all features f by f^*, roles R by R^*, and concepts C by C^*, and extend this notation in the obvious way to apply to inclusion dependencies and terminologies.

Definition 2 (Ordering Description).
Let D be an $\mathcal{ALCQ}(\mathcal{D})$ concept description, f a concrete feature, and R a role. An ordering description *is defined by the grammar:*

$$Od \quad ::= \quad \text{Un} \;\mid\; f : Od \;\mid\; D(Od, Od) \;\mid\; D(Od, Od] \;\mid\; R.f : Od \;\mid\; \sqsubseteq$$

An instance of the first constructor is called the null ordering. *The second constructor is called a* feature value ordering. *The third and forth constructors are called a strong and weak* partition ordering *respectively. The fifth constructor is called a* role nested ordering. *The final constructor is called a* subsumption ordering.

For a given terminology $\mathcal{T}$ and concept descriptions D and E, D is ordered before *E by ordering description Od with respect to $\mathcal{T}$, denoted $(Od)_{\mathcal{T}}(D, E)$, if $\mathcal{T} \not\models D \sqsubseteq \bot$, $\mathcal{T} \not\models E \sqsubseteq \bot$, and at least one of the following conditions holds:*

- $Od =$ "$f : Od_1$" *and* $(\mathcal{T} \cup \mathcal{T}^*) \models (D \sqcap E^*) \sqsubseteq (f < f^*)$,
- $Od =$ "$f : Od_1$" *and* $(Od_1)_{\mathcal{T}}(D, E)$ *and* $(\mathcal{T} \cup \mathcal{T}^*) \models (D \sqcap E^*) \sqsubseteq (f = f^*)$,
- $Od =$ "$D_1(Od_1, Od_2)$" *and* $\mathcal{T} \models D \sqsubseteq D_1$ *and* $\mathcal{T} \models E \sqsubseteq \neg D_1$,
- $Od =$ "$D_1(Od_1, Od_2)$", $(Od_1)_{\mathcal{T}}(D, E)$ *and* $\mathcal{T} \models (D \sqcup E) \sqsubseteq D_1$,
- $Od =$ "$D_1(Od_1, Od_2)$", $(Od_2)_{\mathcal{T}}(D, E)$, $\mathcal{T} \models D \sqsubseteq \neg D_1$, *and* $\mathcal{T} \models E \sqsubseteq \neg D_1$,
- $Od =$ "$D_1(Od_1, Od_2]$" *and* $\mathcal{T} \models D \sqsubseteq D_1$ *and* $\mathcal{T} \not\models E \sqsubseteq D_1$,
- $Od =$ "$D_1(Od_1, Od_2]$", $(Od_1)_{\mathcal{T}}(D, E)$ *and* $\mathcal{T} \models (D \sqcup E) \sqsubseteq D_1$,
- $Od =$ "$D_1(Od_1, Od_2]$", $(Od_2)_{\mathcal{T}}(D, E)$, $\mathcal{T} \not\models D \sqsubseteq D_1$, *and* $\mathcal{T} \not\models E \sqsubseteq D_1$,
- $Od =$ "$R.f : Od_1$" *and there exists k such that* $\mathcal{T} \models D \sqsubseteq \exists R.(f \leq k) \sqcap \forall R.(f \leq k)$ *and* $\mathcal{T} \models E \sqsubseteq \forall R.(f > k)$,
- $Od =$ "$R.f : Od_1$", $(Od_1)_{\mathcal{T}}(D, E)$, *and there exists k such that* $\mathcal{T} \models D \sqsubseteq \forall R.(f = k)$ *and* $\mathcal{T} \models E \sqsubseteq \forall R.(f = k)$,
- $Od =$ "$\sqsubseteq$", $\mathcal{T} \models D \sqsubseteq E$ *and* $\mathcal{T} \not\models E \sqsubseteq D$.

Two descriptions D and E are said to be incomparable with respect to an ordering Od and terminology $\mathcal{T}$ *if* $\neg(Od)_{\mathcal{T}}(D, E)$ *and* $\neg(Od)_{\mathcal{T}}(E, D)$, *or simply* incomparable *when Od and $\mathcal{T}$ are clear from context.*

Definition 3 (Description Tree). *Let D denote an arbitrary concept description in $\mathcal{ALCQ}(\mathcal{D})$. A* description tree *is an ordered rooted binary tree conforming to the grammar:*

$$Tr, L, R \quad ::= \quad \langle\rangle \quad | \quad \langle D, L, R\rangle.$$

An instance of the first production denotes an empty tree, while an instance of the second production denotes a node at the root of a tree with left subtree L, right subtree R, and labeled *by D. We write $\langle D, L, R\rangle \in Tr$ if $\langle D, L, R\rangle$ is a node occurring in Tr, and call any tree of the form $\langle D, \langle\rangle, \langle\rangle\rangle$ a* leaf *node.*

A description tree Tr is well formed for ordering description Od with respect to terminology $\mathcal{T}$ *if, for all $\langle D, L, R\rangle \in Tr$,*

- $\mathcal{T} \not\models D \sqsubseteq \bot$,
- $\neg(Od)_{\mathcal{T}}(D, D')$ *for all* $\langle D', L', R'\rangle \in L$, *and*
- $\neg(Od)_{\mathcal{T}}(D', D)$ *for all* $\langle D', L', R'\rangle \in R$.

When Od and $\mathcal{T}$ are clear from context, we say simply that Tr is well formed.

For a given ordering description Od, the conditions for Tr to be well formed provide the invariants for insertions of new nodes. For example, when inserting a new node for description D' in description tree $\langle D, L, R\rangle$, a new leaf node $\langle D', \langle\rangle, \langle\rangle\rangle$ must be added in subtree L if $(Od)_{\mathcal{T}}(D', D)$.

Definition 4 (Description Index). *Let $\mathcal{T}$ be a terminology, Od an ordering description, and Tr a well formed description tree with respect to Od and $\mathcal{T}$. A* description index *is a 3-tuple $\langle Tr, Od, \mathcal{T}\rangle$.*

We consider queries Q of the form $\langle D_Q, Od_Q\rangle$, where D_Q is a concept in $\mathcal{ALCQ}(\mathcal{D})$ and Od_Q is an ordering description. A user presumes that query Q is evaluated with respect to an index $\langle Tr, Od, \mathcal{T}\rangle$ by first finding all concepts E_i labelling nodes in Tr for which $\mathcal{T} \models E_i \sqsubseteq D_Q$ and then sorting the concepts E_i according to Od_Q.

2.1 Properties

The original presentation of description indices in [3] defined properties of the indices (and the associated ordering descriptions) that collectively allowed efficient search and order optimization to be performed. To provide a thorough analysis of the proposed ordering descriptions, and the behaviour of the ordering descriptions in the presence of different classes of data descriptions, we first separate the definitions of the properties to identify the functionality they enable. This creates a framework for general discussion of orderings and their properties.

To begin, the following property establishes that an ordering description is irreflexive, asymmetric, and transitive. An ordering description satisfying this property would therefore define a strict partial order over concept descriptions.

Property 1 (Partial Order) *Given a terminology $\mathcal{T}$, ordering description Od, and concept descriptions D_1, D_2, and D_3:*

1. $\neg(Od)_{\mathcal{T}}(D_1, D_1)$;
2. *If* $(Od)_{\mathcal{T}}(D_1, D_2)$, *then* $\neg(Od)_{\mathcal{T}}(D_2, D_1)$;
3. *If* $(Od)_{\mathcal{T}}(D_1, D_2)$ *and* $(Od)_{\mathcal{T}}(D_2, D_3)$; *then* $(Od)_{\mathcal{T}}(D_1, D_3)$.

Counting The following Property does not have a direct impact on the performance of description indices, but can be a useful property if one wishes to extend the query capabilities to include a count aggregate (that is, a count of objects denoted by descriptions in a query result). The following property guarantees disjointness between orderable descriptions.

Property 2 (Disjointness) *Given a terminology* $\mathcal{T}$, *ordering description* Od, *and concept descriptions* $D_1 and D_2$:

$$\text{If } (Od)_{\mathcal{T}}(D_1, D_2), \text{ then } \mathcal{T} \models (D_1 \sqcap D_2) \sqsubseteq \bot.$$

Pruning The following two properties describe an important feature of ordering descriptions that enables pruning in description indices during search. Note that not all ordering constructors satisfy both of these properties (see Section 2.2).

Property 3 (Left Pruning) *Given a terminology* $\mathcal{T}$, *ordering description* Od, *and concept descriptions* D_1, D_2, *and* D_3:

$$\text{If } (Od)_{\mathcal{T}}(D_1, D_2), \mathcal{T} \models D_3 \sqsubseteq D_2 \text{ and } \mathcal{T} \not\models D_3 \sqsubseteq \bot, \text{ then } (Od)_{\mathcal{T}}(D_1, D_3).$$

Property 4 (Right Pruning) *Given a terminology* $\mathcal{T}$, *ordering description* Od, *and concept descriptions* D_1, D_2, *and* D_3:

$$\text{If } (Od)_{\mathcal{T}}(D_1, D_2), \mathcal{T} \models D_3 \sqsubseteq D_1 \text{ and } \mathcal{T} \not\models D_3 \sqsubseteq \bot, \text{ then } (Od)_{\mathcal{T}}(D_3, D_2).$$

Descriptive Sufficiency In some cases, in order to guarantee that rotations and order optimization can be performed, we need to introduce a limitation on the types of descriptions that are being indexed. This limitation ensures, for example, that descriptions supply values for indexed concrete features, and are partitionable by the partition orderings. We call this property *descriptive sufficiency*. In Section 2.2 we will consider the properties of ordering descriptions used to index data in the absence and presence of descriptive sufficiency.

Definition 5 (Descriptive Sufficiency). *A concept description* D *is* sufficiently descriptive for ordering description Od with respect to terminology $\mathcal{T}$, *written* $SD_{\mathcal{T}}(D, Od)$, *if at least one of the following conditions hold:*

- $Od =$ "Un",
- $Od =$ "$f : Od_1$", $SD_{\mathcal{T}}(D, Od_1)$, *and* $\mathcal{T} \models D \sqsubseteq (f = k)$,
- $Od =$ "$R.f : Od_1$", $SD_{\mathcal{T}}(D, Od_1)$, *and* $\mathcal{T} \models D \sqsubseteq \forall R.(f = k)$,
- $Od =$ "$D'(Od_1, Od_2)$", $SD_{\mathcal{T}}(D, Od_1)$, *and* $\mathcal{T} \models D \sqsubseteq D'$,
- $Od =$ "$D'(Od_1, Od_2)$", $SD_{\mathcal{T}}(D, Od_2)$, *and* $\mathcal{T} \models D \sqsubseteq \neg D'$,

for some $k \in \Delta_C$. *When* Od *and* $\mathcal{T}$ *are clear from context, we say simply that* D *is sufficiently descriptive.*

Rotations In order to guarantee efficient search capabilities, description indices need to be able to have rotations performed to ensure a balanced tree is maintained after insertions. The following property establishes that both left and right tree rotations can be performed on description indices without violating the well formedness property of the tree.

Property 5 (Tree Rotation) *Given an ordering description Od, terminology $\mathcal{T}$, and concept descriptions D_1 and D_2, for any description trees Tr_1, Tr_2, and Tr_3 that are well formed, $\langle D_1, \langle D_2, Tr_1, Tr_2\rangle, Tr_3\rangle$ is well formed if and only if $\langle D_2, Tr_1, \langle D_1, Tr_2, Tr_3\rangle\rangle$ is well formed.*

Order Optimization The last property of description indices that we are interested in, order optimization, is the ability to avoid sorting a query result when the order in which the indexed descriptions are retrieved is already consistent with the order specified by the query. This property is given by the refinement relationship. A sound procedure for computing refinement can be found in [3].

Definition 6 (Order Refinement). *Given a terminology $\mathcal{T}$, concept description D, and pair of ordering descriptions Od_1 and Od_2, Od_1* refines *Od_2* with respect to $\mathcal{T}$ and D, *written $Od_1 \prec_{\mathcal{T},D} Od_2$, if, for all concept descriptions E_1 and E_2 such that $\mathcal{T} \models (E_1 \sqcup E_2) \sqsubseteq D$:*

$$(Od_2)_{\mathcal{T}}(E_1, E_2) \quad \textit{implies} \quad (Od_1)_{\mathcal{T}}(E_1, E_2).$$

Od_1 is equivalent to *Od_2* with respect to $\mathcal{T}$ and D, *written $Od_1 \approx_{\mathcal{T},D} Od_2$, when $Od_1 \prec_{\mathcal{T},D} Od_2$ and $Od_2 \prec_{\mathcal{T},D} Od_1$. In all cases, D is called a* parameter description.

Property 6 (Order Optimization) *Given a terminology $\mathcal{T}$, description index $\langle Tr, Od_I, \mathcal{T}\rangle$, and query $\langle D, Od_Q\rangle$ such that $Od_I \prec_{\mathcal{T},D} Od_Q$: $(Od_Q)_{\mathcal{T}}(E_1, E_2)$ for any descriptions E_1 and E_2 occurring in Tr for which E_1 precedes E_2 according to an in-order traversal of Tr.*

2.2 Analysis of Description Indices

We begin by making a few observations about the properties of ordering descriptions as they relate to description indices. The first observation is that all ordering descriptions define partial orders over the space of possible concept descriptions.

Observation Given a terminology $\mathcal{T}$, all possible ordering descriptions Od satisfy Property 1 with respect to $\mathcal{T}$.

The second observation extends pruning, Property 3 and Property 4, to description indices by the nature of well formed trees. Because description indices have well formed trees by definition, this observation holds for any description

In Absence of Descriptive Sufficiency

	Disjoint	Prune Left	Prune Right	Rotate	Order Opt.
Un	–	–	–	✓	✓
$f : Od$	✓	✓	✓	×	×
$D(Od_1, Od_2)$	✓	✓	✓	×	×
$D(Od_1, Od_2]$	×	×	✓	✓	✓
$R.f : Od$	✓	×	×	×	×
$\sqsubseteq$	×	×	✓	✓	✓

With Descriptive Sufficiency

	Disjoint	Prune Left	Prune Right	Rotate	Order Opt.
Un	–	–	–	✓	✓
$f : Od$	✓	✓	✓	✓	✓
$D(Od_1, Od_2)$	✓	✓	✓	✓	✓
$R.f : Od$	✓	✓	✓	✓	✓

Table 1. Properties of Ordering Descriptions

index with an ordering description satisfying Property 3 and Property 4 for part one and two of the observation respectively.

Observation For any description index $\langle Tr, Od, \mathcal{T} \rangle$, node $\langle D, L, R \rangle \in Tr$, and concept description E:

1. if Od satisfies Property 3 then $(Od)_{\mathcal{T}}(D, E)$ implies $\mathcal{T} \not\models D' \sqsubseteq E$ for any node $\langle D', L', R' \rangle \in L$, and
2. if Od satisfies Property 4 then $(Od)_{\mathcal{T}}(E, D)$ implies $\mathcal{T} \not\models D' \sqsubseteq E$ for any node $\langle D', L', R' \rangle \in R$.

The properties of ordering descriptions are summarized in Table 1. The table illustrates the properties of each ordering constructor in the absence and presence of descriptive sufficiency. An arbitrary ordering description thus has only the properties that are shared by every construct used in the ordering description. As illustrated in the tables, it is not always the case that pruning can be performed for both left and right subtrees, meaning logarithmic tree traversal cannot be guaranteed in all cases. Similarly, not enforcing descriptive sufficiency allows us to index a wider class of data descriptions, but in many cases at the cost of rotations and order optimization. Thus, we cannot ensure a balanced tree and may potentially have to sort a result to satisfy the query specification. By enforcing descriptive sufficiency we lose the weak partition ordering and subsumption ordering constructors, but gain the full set of properties for all other ordering constructors. Note that the "–" symbol denotes a non-applicable field for the "Un" operator since it is by definition, always $false$.

XQuery

```
for $item in /catalog/item
where ($item/author/mailing_address/name_of_state ="New York"
or $item/publisher/mailing_address/name_of_state="New York" )
and $item/date_of_release gt "1995-01-01"
and $item/date_of_release lt "2005-01-01"
return $item
```

Concept Description

$ITEM \sqcap$ (date_of_release > 1995-01-01) $\sqcap$ (date_of_release < 2005-01-01) $\sqcap$ ($\exists hasAuthor.(\exists hasMailingAddress.$(name_of_state = "New York")) $\sqcup \exists hasPublisher.(\exists hasMailingAddress.$(name_of_state = "New York")))

Fig. 1. Example XQuery and Associated Concept Description

3 Experimental Evaluation

In order to demonstrate the feasibility and potential benefit of our approach, we built a prototype implementation of description indices. The implementation uses off-the-shelf open-source tools along with a small Java core to link them. We use the FaCT++ description logic reasoner [4] to perform the subsumption testing, the DIG XML interface [5] for concept description representation, and the Xerces XML library [6] for data parsing. Communication with FaCT++ is via a self-hosted HTTP connection.

We used XBench [7], an XML benchmark as the basis for our experiments. The goal of the evaluation was to compare the performance of our tree-based description indices to a traditional tree-based indexing method. We use the X-Hive XML database [8] as a representative XML engine with indexing capabilities, and also include Qexo [9] and Galax [10], two popular streaming XML query processors (no indexing) for reference.

We use a mapping from XML entities to $\mathcal{ALCQ}(\mathcal{D})$ concept descriptions that is a simple conversion preserving the semantics of the raw data and XML structure. Nested entities are modeled with role relations, and data nodes an attributes are modeled with concrete features. Similarly, we map the XQueries from the benchmark into a description and ordering pair. Because of the simplicity of this model, we can only support queries which are expressible as a concept description, and thus cannot handle constructive queries like joins. Also, our model does not have the capacity to express projections, so the results from our system are always the top level entities being indexed. Figure 1 shows a sample XQuery (labeled as Q21 in our experiments) and the concept description translation (note that long XML paths are simplified for illustrative purposes). The query finds all item entities released during a certain time period that have either an author or publisher from New York. The XML data itself is translated to concept descriptions in an analogous way, mapping all data items and structural components to concrete features and roles respectively.

	Our System	X-Hive	Qexo		Galax	
	Query Time	Query Time	Total Time	Adj. Time	Total Time	Adj. Time
Q_1	7	**4**	2652	1680	4373	3401
Q_2	1164	**1006**	2009	1037	3740	2768
Q_5	**8**	9	1664	692	3591	2619
Q_6	**22**	915	2012	1040	3907	2935
Q_8	**3**	422	1668	696	3580	2608
Q_9	**2**	4	1664	692	3603	2631
Q_{12}	**2**	69	1672	700	3550	2578
Q_{14}	**7**	701	1720	748	3612	2640
Q_{21}	**439**	9332	3910	2938	9367	8395
Q_{22}	**121**	522	3160	2188	N/A	N/A

Table 2. Query processing run times (msec).

3.1 Experimental Setup

The experiments were run on a Linux based 1.66 GHz dual-core system, with 1 GB of main memory. We used the data-centric single document benchmark (DC/SD) from the XBench suite [7], which contains a synthetic XML document with publication data.

We consider data generated in three sizes, the first with 2,500 items (approximately 10MB), the second with 13,750 items (approximately 55MB), and the last with 25,000 items (approximately 100MB). We use eight queries from the XBench DC/SD workload that are expressible as concept descriptions and two additional queries that illustrate the advantage of our proposed enhancements.

The first query, labeled as query Q_{21}, is supported by it's associated index, and takes advantage of the partition ordering of ordering descriptions. The second query, labeled as query Q_{22}, is not supported by an index, but is the only query containing a disjunction to illustrate the utility of using the DL reasoner.

Both our system and the X-Hive system [8] preprocess and index the XML data before query processing. We manually create the appropriate indices in both systems to maximize the performance of each query. This entails creating the best set of XML indices (determined by experimentation) for X-Hive, and the appropriate ordering description for a description index in our system. We consider the fragment of our ordering language that retains the full set of indexing properties as shown in Table 1. The Qexo [9] and Galax [10] systems, however, are file streaming XQuery engines. As such, they do not have a preprocessing and indexing phase. Because file loading is done during query processing in these systems, we provide a total time and an adjusted time. The adjusted time is calculated by subtracting a constant factor (determined by experimentation) to account for the average file loading time and depends on the size of the file.

Number of Items	2500	13750	25000
Our System (Q_1)	7	11	**120**
X-Hive (Q_1)	**4**	**10**	330
Qexo (Q_1)	1680	4348	6357
Galax (Q_1)	3401	34712	97095
Our System (Q_6)	**22**	**117**	**198**
X-Hive (Q_6)	915	3838	7001
Qexo (Q_6)	1040	4111	5597
Galax (Q_6)	2935	33126	94976

Table 3. Comparison times for all three data sets (msec).

3.2 Results

The experimental results for the 2,500 item data set, shown in Table 2, demonstrate that our implementation is comparable with the other three systems. (Note that the numbers for the first eight queries correspond to the numbering from the XBench benchmark.) The table shows query processing times for our system and X-Hive, and total run time and adjusted times for Qexo and Galax as previously described. We outperform the other systems by a significant margin on queries 6, 8, and 14 of the XBench benchmark because we are able to exploit description indices for the queries. Conversely, X-Hive, the only other system that creates indices, does not support index structures that are expressive enough to efficiently answer these queries. Our system suffers on query 2 because of the HTTP and FaCT++ overhead. In particular, this query requires a complete scan of the data set resulting in a large amount of data transfer for subsumption testing.

Our supplied query 21 forces a partitioning of the data, followed by two independent sorts. Because this construct can be captured by our ordering descriptions, we can create an index that supports the query and avoid all of the required tasks by simply retrieving the data in the desired order. Conversely, The XQuery engines are forced to perform the partition and sort operations, causing a significant discrepancy in performance.

Query 22 is the only query that contains a disjunction, which we suspect is harder for XQuery processors to handle. We attribute the good performance of our system for this query to the efficiency of FaCT++ in computing if a concept description qualifies as a query result.

For the remainder of the queries, our description indices are similar to the indices created by X-Hive, and consequently have comparable performance. These situations correspond to indices with simple feature value orderings. The difference is that we use FaCT++ to check if candidate results qualify, while X-Hive traverses the XML to find all relevant values needed to evaluate the predicate.

Table 3 shows the result of running Queries 1 and 6 on all three data sets. Query 1 is taken as representative query in which both our system and the X-Hive system can use an index. Query 6 on the other hand, represents a situation in

which our system can exploit a description index, while X-Hive cannot. Because our system and X-Hive use indices for query 1, and the other two systems do not, our system and X-Hive scale much better than the other two systems. Query 6, the case in which ours is the only system that is able to use an index, shows that our system still scales well with the partitioning ordering description, while the other systems are forced to perform a linear scan of the data. This is a promising result, since it shows that the potentially complex subsumption calls to FaCT++ during index traversal do not have a substantial impact on performance.

4 Summary and Discussion

We have explored the properties of ordering descriptions, including some new ordering constructors, in different classes of data descriptions. Our results show that one can impose varying levels of restrictions on the descriptions being indexed in order to achieve the desired index properties. This allows flexibility in applying description indices to particular problems.

Our experimental results suggest that enabling potentially complex subsumption tests during query evaluation has a tolerable overhead. While we acknowledge that the XML example is a rather simplistic data set, lacking the worst case scenarios of DL reasoning, we have found that other DL expressible data sets with large terminologies, such as YAGO [11], share similar properties with our XML example (i.e. entities described by mostly conjunctive descriptions). Thus, we feel our indexing method can play a pivotal role in enhancing technologies such as ABox querying and semantic search.

References

1. Resource Description Framework (RDF): http://www.w3.org/RDF/
2. Web Ontology Language (OWL): http://www.w3.org/2004/OWL/
3. Pound, J., Stanchev, L., Toman, D., Weddell, G.: On Ordering Descriptions in a Description Logic. Proc. 20th Int. Workshop on Description Logics **250** (2007) 123–134
4. Dmitry Tsarkov and Ian Horrocks: FaCT++ Description Logic Reasoner: System Description . In: 3rd International Joint Conference on Automated Reasoning. (2006)
5. Bechhofer, S., Moller, R., Crowther, P.: The DIG Description Logic Interface. In: In Proc. of International Workshop on Description Logics (DL2003). (2003)
6. Xerces XML Parser: http://xerces.apache.org/xerces-j/
7. B. B. Yao, M. T. Ozsu, and N. Khandelwal: XBench Benchmark and Performance Testing of XML DBMSs. In: IEEE International Conference on Data Engineering. (2004) 621–632
8. X-Hive/DB: `http://www.x-hive.com`
9. Qexo: `http://www.gnu.org/software/qexo/`
10. Galax: `http://www.galaxquery.org/`
11. Suchanek, F.M., Kasneci, G., Weikum, G.: Yago: A Core of Semantic Knowledge - Unifying WordNet and Wikipedia. In: 16th International World Wide Web Conference (WWW 2007), Banff, Canada, ACM (2007) 697–706