Toward Expressive Syndication on the Web

Christian Halaschek-Wiener
Department of Computer Science
University of Maryland
College Park, Maryland
halasche@cs.umd.edu

James Hendler
Department of Computer Science
University of Maryland
College Park, Maryland
hendler@cs.umd.edu

ABSTRACT

Syndication systems on the Web have attracted vast amounts of attention in recent years. As technologies have emerged and matured, there has been a transition to more expressive syndication approaches; that is, subscribers and publishers are provided with more expressive means of describing their interests and published content, enabling more accurate information filtering. In this paper, we formalize a syndication architecture that utilizes expressive Web ontologies and logic-based reasoning for selective content dissemination. This provides finer grained control for filtering and automated reasoning for discovering implicit subscription matches, both of which are not achievable in less expressive approaches. We then address one of the main limitations with such a syndication approach, namely matching newly published information with subscription requests in an efficient and practical manner. To this end, we investigate continuous query answering for a large subset of the Web Ontology Language (OWL); specifically, we formally define continuous queries for OWL knowledge bases and present a novel algorithm for continuous query answering in a large subset of this language. Lastly, an evaluation of the query approach is shown, demonstrating its effectiveness for syndication purposes.

Categories and Subject Descriptors

I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval-Information Filtering

General Terms

Algorithms, Design, Performance

Keywords

Syndication, Publish/Subscribe, Description Logics, Continuous Query Answering

1. INTRODUCTION

Web-based syndication systems have attracted a great amount of attention in recent years. In typical syndication frameworks, users register their subscription requests with syndication brokers; similarly, content publishers register their feeds with syndication brokers. It is then the broker's task to match newly published information with registered subscriptions. As technologies have emerged and matured, there has been a transition to more expressive syndication approaches; that is, subscribers and publishers are provided with more expressive means for describing their interests and published content, enabling more accurate dissemination. Through the years there has been a transition from keyword-based approaches (e.g., [19]) to attribute-value pairs (e.g., [1]) and more recently to XML (e.g., [4]). Given the limited knowledge modeling expressivity of XML (and XML Schema), there has been interest in using RDF for syndication purposes (e.g., [23]). RDF has even been adopted as the standard representation format of RSS 1.0.

Today's syndication approaches still provide relatively weak expressive power from a modeling perspective (i.e., XML and RDF are inexpressive modeling languages) and provide very little automated reasoning support. However, if a more expressive approach with a formal semantics can be provided, many benefits can be achieved; these include a rich semantics-based mechanism for expressing subscriptions and published content, allowing increased selectivity and finer grained control for filtering [22]. Additionally, reasoning can be utilized for discovering subscription matches not found using traditional syntactic syndication approaches.

In this work, we consider using the Web Ontology Language (OWL) for representing published content. As the semantics of a large subset of OWL is aligned with description logics (DLs), reasoning techniques for DLs can then be leveraged for matching content with subscription requests [9, 22, 17]. In such an approach, the previously mentioned benefits of using a formal representation language can therefore be achieved. An additional benefit of an OWL-based syndication approach is its native Web embedding and power as a data integration language. Further, such an approach can be seen as a natural extension of existing RSS 1.0 syndication systems, as OWL can be encoded in RDF.

To demonstrate the increased expressivity of an OWL-based syndication approach, consider the following example related to the financial domain. Assume that a stock trader is interested in information contained in news articles (or collections of articles) that discuss news about companies that will make their stocks volatile (i.e., they become risky investments). In particular, assume that the trader is interested in any *RiskyCompany* that he or she defines to include companies that had their credit downgraded by either `Moodys` or `S&P` credit agency and exists on some sell ratings list of a financial institution. Using an XML-based approach, syndication brokers can provide an XML Schema that contains an element *RiskyCompany* and such companies can be declared to be this type of element. However, more complex logical definitions and automatic classification of objects cannot be supported; therefore, an XML-based approach cannot accommodate the previous example. If we consider an RDF-based approach, then a syndication broker can model the financial domain using RDF Schema;

WWW 2007, May 8–12, 2007, Banff, Alberta, Canada.
ACM 978-1-59593-654-7/07/0005.

therefore, slightly more complex subscription matches can be obtained, as one can logically infer that a company is a *RiskyCompany* (e.g., based on subclass relationships). However, in an RDF-based approach, complex logical definitions, such as the previously mentioned *RiskyCompany*, are not definable. In contrast, in an OWL-based approach, such expressivity is easily provided. For example, *RiskyCompany* is defined in Table 1 (turtle syntax).

```
:RiskyCompany a owl:Class;
  owl:intersectionOf (
    [ a owl:Restriction; owl:onProperty :onRecommendation;
      owl:someValuesFrom :SellList ]
    [ a owl:Restriction; owl:onProperty :downgradedBy;
      owl:someValuesFrom [ owl:oneOf ( :SandP :Moodys ) ] ]
    :Company
  ) .
:movedToJunk a owl:ObjectProperty;
  rdfs:subPropertyOf :downgradedBy .
:onRecommendation a owl:ObjectProperty;
  owl:InverseOf :hasRecommendation .
```

Table 1: Illustration of expressivity in OWL-based syndication.

The definitions that the property *movedToJunk* is a sub-property of *downgradedBy*, and *onRecommendation* is the inverse of *hasRecommentation*, are included in Table 1 as they are used later in the paper.

While OWL-based syndication approaches provide increased expressivity over XML and RDF, previous DL-based syndication approaches suffer from scalability issues due to the inherent complexity of DL reasoning [22, 17, 9]. This is an issue in domains such as the syndication of financial news feeds because response times must be minimal as critical information must be delivered in near real time (e.g., for stock trading purposes). One of the main limitations is related to DL reasoning over changing data; this is primarily due to the static nature of existing DL reasoning techniques. In particular, the addition of information from newly published documents and data can be viewed as a change in the underlying knowledge base (KB). In current DL reasoning algorithms, reasoning on the updated KB is performed from scratch. The consistency of the KB must be ensured; queries must be re-evalutated; etc. This negatively impacts the performance results of existing DL-based syndication approaches, as performance times are in the tens of seconds. An additional limitation of OWL-based syndication approaches is related to the infancy of the underlying architectures investigated to date; in particular, these architectures have not been investigated in great depth or fully formalized.

In this paper we address both of the previously discussed shortcomings with OWL-based syndication systems. First, we formalize a DL-based syndication framework. We then address the scalability of DL reasoning for the purpose of syndication, by first presenting a technique for incremental consistency checking for a substantial portion of OWL. Then, we address continuous query answering over OWL KBs that are being updated, primarily focusing on reducing the size of the KB that must be considered as candidate query bindings. This effectively allows a smaller subset of the KB to be considered for possible subscription matches. The techniques we present are applicable to queries with at least one distinguished variable (i.e., must be bound to a named individual) and containing only simple roles (i.e., no transitive roles or super-roles of a transitive role). Further, the approach supports the description logic $\mathcal{SHI}$ (a large subset of OWL) with the restriction the KB unfoldable (i.e., acyclic). Lastly, an evaluation of the incremental reasoning techniques is provided, demonstrating their effectiveness for OWL-based syndication. Full proofs of the results presented in this work can be found in the accompanying technical report [10].

2. PRELIMINARIES

In this section, we briefly provide an overview of OWL and description logics, query answering for DL KBs, and tableau algorithms for DL reasoning.

2.1 The Web Ontology Language

The W3C-approved Web Ontology Language (OWL) is the recommended standard for the formally representing content on the Web. One of the main benefits of OWL is the support for formal reasoning, as the semantics of a variety of its sub-languages are firmly founded in description logics (a decidable fragment of First Order Logic). In particular, the sub-language OWL-DL is a syntactic variant of the description logic $\mathcal{SHOIN}$ [13], with an OWL-DL ontology corresponding to a $\mathcal{SHOIN}$ KB. In this work, we address a subset of $\mathcal{SHOIN}$, namely $\mathcal{SHI}$; therefore, we briefly introduce the syntax of $\mathcal{SHI}$ (semantics can be found in [13]).

Let $\mathbf{C}, \mathbf{R}, \mathbf{I}$ be non-empty and pair-wise disjoint sets of *atomic concepts*, *atomic roles*, and *individuals* respectively. The set of $\mathcal{SHI}$ roles (roles, for short) is the set $\mathbf{R} \cup \{R^- \mid R \in \mathbf{R}\}$, where R^- denotes the inverse of the atomic role R. Concepts are inductively using the following grammar:

$$C \leftarrow A \mid \neg C \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \exists R.C \mid \forall R.C$$

where $A \in \mathbf{C}$, $a \in \mathbf{I}$, $C_{(i)}$ a $\mathcal{SHI}$ concept, R a role, and S a *simple* role (i.e., no transitive roles or super-roles of a transitive role)[1]. We write $\top$ and $\bot$ to abbreviate $C \sqcup \neg C$ and $C \sqcap \neg C$ respectively.

A *role inclusion axiom* is an expression of the form $R_1 \sqsubseteq R_2$, where R_1, R_2 are roles. A *transitivity axiom* is an expression of the form $\mathsf{Trans}(R)$, where $R \in \mathbf{R}$. An RBox R is a finite set of role inclusion axioms and transitivity axioms. To avoid considering the role R^{--} we introduce the function $\mathsf{Inv}(R)$, which returns the inverse of a role R. Additionally, for a role hierarchy $\mathcal{R}$ let the symbol $\sqsubseteq_{\mathcal{R}}$ denote the reflexive transitive closure of $\sqsubseteq$ on $\mathcal{R} \cup \{\mathsf{Inv}(R_1) \sqsubseteq \mathsf{Inv}(R_2) \mid R_1 \sqsubseteq R_2 \in \mathsf{R}\}$. We also use $R_1 \equiv_{\mathcal{R}} R_2$ as an abbreviation for $R_1 \sqsubseteq_{\mathcal{R}} R_2$ and $R_2 \sqsubseteq_{\mathcal{R}} R_1$. We define the function $Tr(R, \mathcal{R})$ that returns *true* if R is a transitive role; otherwise the function returns *false*. A role R_1 is considered *simple* with respect to R if $Tr(R_2, \mathcal{R}) = \mathit{false}$ for all $R_2 \sqsubseteq_{\mathcal{R}} R_1$.

For C, D concepts, a *concept inclusion axiom* is an expression of the form $C \sqsubseteq D$. A TBox T is a finite set of concept inclusion axioms. An ABox A is a finite set of concept assertions of the form $C(a)$ (where C can be an arbitrary concept expression), role assertions of the form $R(a, b)$ and inequality (equality) assertions of the form $a \neq b$ (respectively $a = b$). A KB $\mathsf{K} = (\mathsf{T}, \mathsf{R}, \mathsf{A})$ is composed of TBox T, RBox R and ABox A. Denote the set of individuals in KB K (ABox assertion α) as $\mathbf{I}_{\mathsf{K}}$ (respectively $\mathbf{I}_{\alpha}$).

We also introduce the following notation: denote by $Mod(\mathsf{K})$ the set of all models for K. Additionally, given a $\mathcal{SHI}$ concept C, denote by $Depth(C)$ the maximum modal depth for C (i.e., the maximum nesting depth of quantifiers). Lastly, we provide a brief overview of conjunctive ABox queries (query, for short) for description logics. A query Q consists of a conjunction of ABox assertions of the form $C(a)$ or $R(a, b)$ (see [14] for a precise definition), in which variables can be used in place of individuals and are considered as existentially quantified (the set of variable names, denoted $V(Q)$, is assumed to be distinct from the individual names, $\mathbf{I}$). Query answering is the task for determining if Q is a logical consequence of the KB K (denoted $\mathsf{K} \models Q$); that is, determining if for all models $\mathcal{I}$ of K, $\mathcal{I} \models Q$. As query retrieval is addressed in this work, we briefly introduce the following notation (adopted from [14]): $\langle x_1, ..., x_n \rangle \leftarrow Q$ indicates that the variables $x_1, ...x_n$

[1]See [13] for a precise definition of simple roles.

appearing in Q must be bound to individual names, therefore constituting the answer to the query. These variables are referred to as *distinguished* variables, denoted $DV(Q)$. The *answer set* of a query $\langle x_1, ..., x_n \rangle \leftarrow Q$ w.r.t. to K is the set n-ary tuples defined by the following:

$$\{\langle a_1, ..., a_n \rangle \in \mathbf{I}^n_{\mathsf{K}} \mid \mathsf{K} \models Q[x_1/a_1, ..., x_n/a_n]\}$$

where $Q[x/a]$ represents the query, Q, with all occurrences of variable x substituted by the individual name a. Lastly, we note that if a query can be partitioned into unconnected components (i.e., components that do not share variables), then they are considered independently. Without loss of generality, we assume queries are connected in the remainder of this work [7]. We additionally introduce the following notation: given query Q, let $Con(Q)$, $Rol(Q)$ denote the set of concepts and roles in Q respectively. In the remainder, we assume queries contain at least one distinguished variable; thus, the query can be rolled-up [14] into a distinguished variable. Given a query Q, with abuse of notation, we denote by $Depth(Q)$ the maximum $Depth(C)$, where C is the query rolled-up into a distinguished variable.

2.2 Tableau Algorithms

DL tableau-based algorithms decide the consistency of an ABox A with respect to a TBox T and RBox R by trying to construct (an abstraction of) a common model for A, T, and R, called a *completion graph* [13]. Each node in the graph represents an individual that is labeled with a set of concepts that it satisfies (in the particular model). Formally, a completion graph for an ABox A with respect to T is a directed graph $G = (\mathcal{V}, \mathcal{E}, \mathcal{L}, \doteq)$. Each node $x \in \mathcal{V}$ is labeled with a set of concepts $\mathcal{L}(x)$, and each edge $e = \langle x, y \rangle$ with a set $\mathcal{L}(e)$ of role names. The binary predicate $\doteq$ is used for recording inequalities between nodes. This graph is constructed by repeatedly applying a set of tableau *expansion rules*, adding new concept labels and edges to the graph when necessary. This process continues until the tableau is fully expanded and no additional rules can be applied. A node, x, contains a clash if a contradiction exists in its label (e.g., $C, \neg C \in \mathcal{L}(x)$) or between two nodes (equality and/or inequality). It is noted that the tableau algorithm can be saturated such that all possible completions of a KB are found (corresponding to all models). We lastly introduce the following notation: denote by $Comp(\mathsf{K})$ the set of all complete, clash-free completions of K (i.e., all models); additionally, given completion graph G, denote by $Roots(G)$ the subset of G containing root nodes and their labels, as well edges and edge labels between root nodes.

3. SYNDICATION FRAMEWORK

In this section we formally define the generic DL-based syndication framework proposed in this work. As in typical syndication systems, we assume syndication brokers deliver relevant information to the appropriate subscription requests. Within this framework, a subscription is comprised of a conjunctive ABox (instance) query, which represents the subscribers interests, and an expiration time (i.e., the number of time units that the subscription is valid). The subscription query can be thought of as a continuous conjunctive query that should be evaluated until the expiration time. Therefore, the query is issued once over a changing ABox whose results set is continuously updated as the ABox changes. Intuitively, the answer set of a continuous conjunctive ABox query at time t is the set of all variable bindings entailed by the KB at time t and can be seen as an extension of the definition of a conjunctive query presented earlier.

Definition 1. (Continuous Conjunctive ABox Query) Define a continuous conjunction ABox query Q_c with respect to a DL KB K_t (K at time t) such that it produces results at time t, denoted $Q_{c_r}(t)$, as follows:

$$Q_{c_r}(t) = \{\langle a_1, ..., a_n \rangle \in \mathbf{I}^n_{\mathsf{K}_t} \mid \mathsf{K}_t \models Q_c[x_1/a_1, ..., x_n/a_n]\}$$

Given this, a *subscription* is then defined as follows:

Definition 2. (Subscription) A subscription S is defined as a pair (Q_c, t), where Q_c is a continuous conjunctive ABox query that is evaluated for t time units.

We denote the continuous query of a subscription as $\mathsf{S}(Q_c)$ and the expiration time as $\mathsf{S}(t)$. We now define a *subscriber* to be composed of a set of subscriptions and a unique identifier:

Definition 3. (Subscriber) A subscriber Sub is defined to be a pair (s, i), where s is a set of subscriptions and i is a unique identifier.

Denote a subscriber's set of subscriptions as $\mathsf{Sub}(s)$, and its identifier as $\mathsf{Sub}(i)$. Next, we define a *publisher* to be identified by a unique identifier:

Definition 4. (Publisher) A publisher Pub is defined as being composed of and identified by a unique identifier i.

Additionally, a *publication* is defined to be composed of a set of ABox assertions, the number of time units that the publication is valid, and the identifier of the publisher that produced the information; after the specified time units have passed, it is assumed that the publication is discarded.

Definition 5. (Publication) A publication P is defined as a tuple (α, t, p), where α is a set of DL ABox assertions that expire after t time units, and p is the identifier of the publisher that produced the publication.

Given a publication P, denote the set of ABox assertions as $\mathsf{P}(\alpha)$, the expiration time as $\mathsf{P}(t)$, etc. Intuitively, a *syndication broker* maintains a local KB, in which newly published information is integrated. Additionally, the syndication broker maintains the currently registered subscribers (that have associated subscriptions) and publishers. This is formally defined as follows:

Definition 6. (Syndication Broker) A syndication broker B is defined as a tuple (S, P, K_l), where S is a set of subscribers, P is a set of publishers, and K_l is the broker's local DL KB.

We denote a syndication broker's subscriptions, publishers, and KB as $\mathsf{B}(S)$, $\mathsf{B}(P)$, and $\mathsf{B}(\mathsf{K}_l)$ respectively. After a new publication is received, it is the broker's task to determine the subscribers for which this new infromation is relevant. Before defining subscription matches, we define a generic *update function* that takes a set of publications and integrates them into the broker's KB. Such a function is necessary for integrating newly published information into a DL KB.

Definition 7. (Update Function) Define the update function $update(\mathsf{K}, P)$ to take as input a DL KB K and a set of publications P and return a new consistent DL KB K' that is the result of updating K with $\mathsf{P}(\alpha)$, for all $\mathsf{P} \in P$.

Observe that this update function is generic, as there are many different ways to interpret the update. Such problems have been studied extensively in literature for updating logical KBs. We do

not impose a particular type of update semantics in the formalization of the syndication framework; rather, we only enforce that the update function result in a consistent KB. This is necessary because if the updated KB is inconsistent, then everything is trivially entailed. In Section 4.1 we define a specific update function, referred to as *syntactic updates*.

Lastly, we define a match for a subscription request. As information (documents) is published from multiple publishers and remains valid in the broker's local KB for varying time, a match for a subscription can actually be a composition of the information from multiple publications; that is, the information provided in multiple publications collectively forms a match for the query. To the authors' knowledge, recent approaches have not investigated such functionality; rather, only information from individually published documents form a match for a given subscription. However, such a capability is beneficial, as information can be considered collectively and form matches not found otherwise.

We additionally distinguish between two types of subscription matches, namely *information matches* and *publication matches*. An information match refers to the individuals bound to the (distinguished) variables of a continuous query representing a subscription; that is, the result returned to the subscriber is actually the query answer rather than the publication(s) responsible for the answer. This type of match aligns with recent work in XML-based syndication literature, in which the actual information is filtered and the query answers are returned to the user [16]. In contrast, a publication match refers to the collection of publications that satisfy a subscription; that is, given an information match for a registered subscription, return all minimal sets of publications that cause this match to occur; this aligns with the task of selective content-based filtering of publications. It is clear that given an information match, there is a corresponding set of publication matches.

The distinction between these two match types is made as additional computation is needed to derive all the minimal sets of publications responsible for an information match. Further, the type of match required is application dependent; for example, in OWL-based syndication of news feeds, it is clear that publication matches are needed. In contrast, in the financial domain, analysts are generally interested with the actual information rather than the documents themselves. If we consider our previous example involving the concept *RiskyCompany*, we can observe that analysts are likely to be more interested in the actual instances of *riskycompany*, rather than the articles that discuss them; this is intuitive, as the actual query answer is the actionable information for their purposes (e.g., stock trading). In this work, we address both of these matches; however, our current evaluation focuses on information matches, leaving the remainder as future work. We now define an *information match*:

Definition 8. (Information Match) Define a tuple of individuals $\langle a_1, ..., a_n \rangle$ to be an information match at broker B for subscription S at time t, if and only if the following holds:

$$\langle a_1, ..., a_n \rangle \in \mathsf{B}(\mathbf{I}^n_{\mathsf{K}_t}) \wedge \mathsf{B}(\mathsf{K}_t) \models \mathsf{S}(Q_c[x_1/a_1, ..., x_n/a_n])$$

Before defining a publication match, we present the notion of *minimal justifications* for an entailment in DLs, which has been formally investigated in literature [15].

Definition 9. (Minimal Justification) [15] Let $\mathsf{K} \models \alpha$, where α is a DL axiom and K a DL KB. A fragment $\mathsf{K}' \subseteq \mathsf{K}$ is a minimal justification for α in K if $\mathsf{K}' \models \alpha$ and $\mathsf{K}'' \not\models \alpha$ for every $\mathsf{K}'' \subset \mathsf{K}'$. Denote the set of minimal justifications for $\mathsf{K} \models \alpha$ as $\mathsf{Just}(\mathsf{K}, \alpha)$.

Now we present the definition of a *publication match* which utilizes minimal justifications to define to the publications responsible for an information match:

Definition 10. (Publication Match) Let $\langle a_1, ..., a_n \rangle$ be an information match I at broker B for subscription S at time t. Let J be the set of minimal justifications for I:

$$J = \mathsf{Just}(\mathsf{B}(\mathsf{K}_t), \mathsf{S}(Q_c[x_1/a_1, ..., x_n/a_n])\})$$

Define a set of publications P to be a publication match at broker B for subscription S at time t if there exists $j \in J$ such that the following holds:

$$\forall \mathsf{P} \in P(\ \exists a \in j \wedge a \in \mathsf{P}(\alpha)\ ) \wedge \forall a \in j(\ \exists \mathsf{P} \in P \wedge a \in \mathsf{P}(\alpha)\ )$$

We conclude this section with a brief example demonstrating a composite match (both information and publication matches), and the framework in general. Assume a syndication broker B is composed of one subscription and two publishers. Additionally, assume that the broker's local KB contains the axioms defined previously in Table 1. The the broker is composed of the following:

$$S = \{S_1\}, \quad P = \{P_1, P_2\}$$
$$\mathsf{K}_l = \{\ \text{:movedToJunk},\ \text{:onRecommendation},\ \text{:RiskyCompany}\ \}$$

Assume subscriber S_1 has registered the following subscription for all instances of the class *RiskyCompany*:

$$(\text{RiskyCompany}(x), \infty) \in S_1(s)$$

where ∞ indicates that the subscription does not expire. Additionally assume that P_1 publishes that `BOASellList` (assumed to be an instance of *SellList*) has a sell recommendation for `Ford` and P_2 publishes `Moodys` moved `Ford` credit to junk status. This is formalized as follows:

$$\mathsf{P}_{P_1} = (\{\ \texttt{:BOASellList}\ \text{:hasRecommendation}\ \texttt{:Ford}\ \}, \infty, 1)$$
$$\mathsf{P}_{P_2} = (\{\ \texttt{:Moodys}\ \text{:movedToJunk}\ \texttt{:Ford}\ \}, \infty, 2)$$

where ∞ indicates that the publications do not expire. For ease of exposition, assume that P_{P_1} and P_{P_2} arrive at the broker at time 1 and 2 respectively, and that the update function expands the explicit ABox assertions in the broker's KB with those contained in the publication (see Section 4.1 for further details). When P_{P_1} arrives at the broker, $\mathsf{P}_{P_1}(\alpha)$ is integrated into $\mathsf{B}(\mathsf{K}_l)$, resulting in a updated broker KB K'. It is obvious that at this time the individual `Ford` will not satisfy the subscription; therefore, there will not be a match for S_1 at time 1. However, when P_{P_2} is published at time 2 and integrated into K', there is a composite publication match $\{\mathsf{P}_{P_1}, \mathsf{P}_{P_2}\}$ and an information match `Ford` for the subscription (due to various OWL inferences).

4. REASONING FOR SYNDICATION

As discussed earlier, the main limitation in the proposed syndication framework is related to DL reasoning through incremental changes to the underlying KB. Therefore, the remainder of this paper addresses the two previously mentioned performance bottlenecks, namely consistency checking and query answering through updates. Before addressing these issues, we present the update function adopted for this work.

4.1 Syntactic Updates

For the task of syndication, we propose an update function that we refer to as *syntactic updates*, which supports syntactic changes of KB assertions. Intuitively, syntactic updates can be described

as an update in which all new assertions are directly added (or removed) to the asserted (base) axioms. For purpose of this work, ABox assertions can take the form of individual equality (e.g., {:Ford owl:sameAs :FordMotorCorp}) and inequality assertions (e.g., {:Moodysowl:differentFrom :SandP}), concept assertions (e.g., {:Ford a :Company}; note that complex concept assertions are possible as well), and role assertions (e.g., {:Ford :downgradedBy :Moodys}). Formally, this is described as follows:

Definition 11. (Syntactic Updates) Let A be the ABox of an initial KB K. Then, under syntactic updates, updating K with an ABox addition (respectively deletion) α, written as $\mathsf{K} + \alpha$ (resp. $\mathsf{K} - \alpha$), results in an updated ABox A' such that $\mathsf{A}' = \mathsf{A} \cup \{\alpha\}$ (resp. $\mathsf{A}' = \mathsf{A} \setminus \{\alpha\}$). Denote by $\mathsf{K} \oplus \alpha$ the syntactic update of K with α.

This type of update is different when compared to related work in update semantics [18] and belief revision [6] for DLs; however, it is clearly applicable to syndication applications. Further, there have been negative results with respect to other candidate update semantics for DL KBs. In particular, [18] shows that the standard (minimal change) model-based update semantics cannot be represented in the DLs considered in this paper. [6] shows that many DLs, including those considered here, cannot satisfy the rationality postulates proposed in the AGM theory of belief revision. It is clear that under syntactic updates, the resulting KB can be inconsistent after an update; however, as required by Definition 7, the update function must result in a consistent KB. For this work, we assume that if the resulting KB is inconsistent, then the newly published information is rejected (i.e., removed from the KB). While discarding the recent publication may not be the ideal course of action in all syndication systems, addressing this issue further is out of the scope of this paper. However, we plan to address this issue in future work and provide some initial insights in Section 6.

4.2 Incremental Consistency Checking

After newly published information is integrated in the broker's KB, consistency must be rechecked. As stated earlier, with large ABoxes, checking consistency introduces substantial overhead. In this case of syndication, this problem is compounded, as the broker's KB will become substantially large because the KB can contain permanent domain knowledge, as well as publications that remain valid for substantial time periods.

To address this issue, we have recently investigated incremental consistency checking in OWL KBs. In particular, in [12] we present an approach for incrementally updating tableau completion graphs under syntactic ABox updates in the description logics $\mathcal{SHIQ}$ and $\mathcal{SHOQ}$ [12], which encompass the portion of OWL-DL addressed later in this work. In [12] the update algorithm adds new (removes existing for deletions) components (edge, nodes, or labels) introduced by the update to a (cached) completion graph from the consistency check prior to the update; after this, standard tableau completion rules are re-fired to ensure that the model is complete. Therefore, the completion graph built prior to the update (e.g., during the initial consistency check) is cached and updated such that if a model exists (i.e, the KB is consistent after the update), a new completion graph will be found. It was observed that updates did not have a large effect on the existing completion graph; therefore, orders of magnitude performance improvements are achieved. Due to space limitations, further details regarding the approach are omitted here; however, they can be found in [12].

4.3 Continuous Query Answering

After guaranteeing consistency of an updated KB, the various subscriptions registered with the broker can be (re)evaluated. In the remainder of this section, we present an approach for more efficient continuous query answering for a subset of OWL-DL, specifically unfoldable $\mathcal{SHI}$. Two restrictions are imposed on the queries supported in the approach, namely that only simple roles (i.e., no transitive roles or super-roles of a transitive role) can be used in the query, and the query must contain at least one distinguished variable (note that more frequently in realistic scenarios, queries contain some number of distinguished variables). These restrictions enable the techniques presented in the following sections; further query answering in the presence of transitive roles is a relatively open problem (however, see [7]). In the following sections we assume that all concepts are in negation normal form (i.e., negation only occurs in front of concept names), and all concepts are fully unfolded such that they are composed of only primitive (base) concepts [3].

Before discussing the overall goal of the approach, we make a few simplistic observations: by montonicity of $\mathcal{SHI}$ (and OWL-DL in general), continuous query answering in the event of ABox additions reduces to determining any new bindings that are entailed by the KB, whereas handling deletions reduces to guaranteeing that previous bindings are still entailed.

4.3.1 Localizing Effects of Updates

When querying very large ABoxes, one of the main problems is that a large number of individuals in the KB must be considered as potential variable bindings. However, we propose that under the types of updates considered in this work, the candidate query bindings can be drastically pruned. A key insight is demonstrated if we consider a simple query such as $\langle x \rangle \leftarrow \text{Company}(x)$. Intuitively, in the event an update is an addition, we would only like to consider affected named individuals not previously bound to x as potential new bindings (i.e., answers); in contrast if the update is a deletion, only individuals previously bound to x that were affected by the update need to be re-checked. Therefore, the main goal of the approach presented here is to localize the named individuals in the KB that are affected by the update in such a way that they may impact the previous query results.

Before discussing this further, we define the notion of *explicitly affected individuals*, which intuitively are the individuals manipulated during the incremental update of all completions for a KB.

Definition 12. (Explicitly Affected Individuals): Given $\mathcal{SHI}$ KB K and ABox update α, define the explicitly affected individuals, denoted $EI(\mathsf{K}, \alpha)$, to be the set of all named individuals $a \in \mathbf{I}_\mathsf{K} \cup \mathbf{I}_\alpha$ such that either:

1. $a \in \mathbf{I}_\alpha$
2. during the update of some $G \in Comp(\mathsf{K})$ with α (as in Section 4.2), a has some label change, or outgoing/ingoing edge that is added/removed with the following constraints
   (a) if the update introduces non-deterministic choices, each completion is saturated
   (b) if a node is reached, then the expansion rules are re-applied to all of the node labels
3. if a clash occurs on node z when updating $G \in Comp(\mathsf{K})$ then for any node z_i s.t. there exists a path $z_0, ..., z_i$ ($i \geq 1$) in G where $z = z_0$, z_k a P-neighbor of z_{k-1} ($1 \leq k \leq i$), $P \sqsubseteq R$ and $\{\forall R.C, \forall R^-.C\} \cap \mathcal{L}(z_k) \neq \emptyset$ ($1 \leq k \leq i$), C and P, R some concept and roles respectively, it is the case that either
   (a) $a = z_i$
   (b) a is reached by reapply the expansion rules to labels of z_i and any subsequently reached node in some $G \in Comp(\mathsf{K} \oplus \alpha)$

Additionally, we introduce the notion of a *root path* between two individuals:

Definition 13. (Root Path): Define there to be a root path of length D between two nodes x and y of a completion graph if they are reachable by at most D edge traversals where:

1. edge direction is ignored
2. successive traversal of edges labeled with roles that are not simple is only counted once
3. if there exists more than one label for an edge, one of which is not a simple role, then the non-simple edge is traversed and condition 2 is assumed

We now define the general notion of *affected individuals* adopted for the purpose of this work; given these individuals, we show that all new (resp. invalidated) bindings for a query can be found.

Definition 14. (Affected Individuals): Given $\mathcal{SHI}$ KB K, conjunctive query Q, and ABox update α, define an individual $a \in \mathbf{I}_\mathsf{K} \cup \mathbf{I}_\alpha$ to be in the set of affected individuals, denoted $AI(\mathsf{K}, \alpha)$, if either:

1. $a \in EI(\mathsf{K}, \alpha)$
2. for some $b \in \mathcal{V}$ s.t. $D \in \mathcal{L}(b)$, D is of the form $\forall R.C$, and b does not have a P-neighbor ($P \sqsubseteq R$), it is the case that there is a root path of at most length $Depth(Q)$ between a and b in some $G \in Comp(\mathsf{K} + \alpha)$ (resp. $G \in Comp(\mathsf{K})$ for deletions)
3. for some $b \in EI(\mathsf{K}, \alpha)$ there is a root path of at most length $Depth(Q)$ between a and b in some $G \in Comp(\mathsf{K} + \alpha)$ (resp. $G \in Comp(\mathsf{K})$ for deletions).

It can be shown that for there to be a new (resp. invalidated) binding after an update, at least one named individual in the binding must be in $AI(\mathsf{K}, \alpha)$.

Proposition 1. *Let K be a $\mathcal{SHI}$ KB, Q a conjunctive query, and α an ABox update. If $\mathsf{K} \not\models Q[x_1/a_1, ..., x_n/a_n]$ and $\mathsf{K} \oplus \alpha \models Q[x_1/a_1, ..., x_n/a_n]$ (resp. $\mathsf{K} \models Q[x_1/a_1, ..., x_n/a_n]$ and $\mathsf{K} \oplus \alpha \not\models Q[x_1/a_1, ..., x_n/a_n]$), then there exists some named individual $b \in \mathbf{I}_\mathsf{K} \cup \mathbf{I}_\alpha$ that is bound to some $y \in V(Q)$ such that $b \in AI(\mathsf{K}, \alpha)$.*

Proposition 1 is intuitive as it states that for there to be a new (resp. invalidated) binding, then there must exist some individual in that binding that either is directly affected by the update in some completion graph or is in the *proximity* of some other individual that was directly affected. We are able to show that the notion of proximity introduced in Definition 14 (conditions 2 and 3) is sufficient (observe that currently we do not take into account the structure of the concepts in the query; however, we plan to address this in future work). More importantly, Proposition 1 implies that in order to find the affected individuals, one can update all $G \in Comp(\mathsf{K})$ and gather the individuals that satisfy the properties provided.

It is clear, however, that incrementally maintaining all completion graphs for a given KB is not practical; further in the presence of a reasonable degree of non-determinism in a KB, saturating the tableau is a very expensive process. To avoid performing a full saturation of the initial KB, we propose building a structure that we refer to as a *summary root graph*.

Definition 15. (Summary Root Graph): Let G be the completion graph built for $\mathcal{SHI}$ KB K by applying all tableau expansion rules to K as normal, however with the following modifications:

1. if a non-deterministic choice is encountered, add all labels in the disjunction to the node without creating a new branch
2. if a clash is encountered, it is ignored

Define the summary root graph S_G as $S_G = Roots(G)$

Observe that condition 2 is required, as adding all labels from a disjunction can obviously introduce clashes; also note that only the structure for root nodes and their edges is kept to reduce memory overhead. It can be seen that the summary root graph does not correspond to a model of the KB; however, the approach guarantees that if a root node or edge between root nodes has a label in some completion graph corresponding to a model for the KB, then that label will be in the label set for that individual in the summary root graph. The aim behind the approach is to use this structure to localize an overestimate of the explicitly affected individuals; an overestimate is acceptable as, in the end, we are trying to find only *candidate* distinguished variable bindings.

Using the summary root graph, an overestimate for $EI(\mathsf{K}, \alpha)$ can be provided; we first note that in the case of ABox *deletions*, we propose using axiom tracing [2, 15, 12] during the application of expansion rules, effectively tracking the asserted axioms responsible for changes to the summary root graph. We also note that there is a simple modification when checking if the expansion rules can be applied to a node (to guarantee completeness). Specifically, node labels are marked when they have had a completion rule applied to them during the overestimate procedure; if the label is not marked, then the expansion rule is applied. Details are omitted here, however they can be found in Table 2 of [10].

Definition 16. (Overestimate of Explicitly Affected Individuals): Let S_G be the summary root graph for $\mathcal{SHI}$ KB K and α an ABox update. Define the overestimate of explicitly affected individuals, denoted $EI_{S_G}(\mathsf{K}, \alpha)$, to be defined by the following procedure:

1. (a) α an addition: add the structure introduced by the update to S_G (as in [12]) and apply the tableau expansion rules to *all* labels of individuals x of S_G such that $x \in \mathbf{I}_\alpha$
   (b) α a deletion: remove all structures from S_G solely dependent on the deleted assertion (determined as in [12]) and apply the expansion rules to all individuals *and* their neighbors if the node was affected by the initial retraction of structures dependent on α
2. apply the expansion rules to *all* labels of individuals (named or un-named) reached by subsequent rule firings
3. use the modifications to the tableau expansion rules in Definition 15 and Table 2 of [10]
4. if a clash is found, then condition 3 of Definition 12 is checked however, the S_G is used rather than all $G \in Comp(\mathsf{K})$.

$EI_{S_G}(\mathsf{K}, \alpha)$ is then composed of all root nodes that are reached during the application of expansion rules, have a label change, or are adjacent to an edge or edge label that changed.

Note that after the application of expansion rules finishes, it is assumed that un-named nodes and their edges are discarded from the summary root graph (which has therefore been updated). Additionally, when the summary root graph is updated during an addition, axioms traces are updated using the same approach as in [12]. It can be shown that after the update, the overestimate of the explicitly affected individuals satisfies the following property:

Proposition 2. *Given a $\mathcal{SHI}$ KB K, ABox update α and summary root graph S_G for K, then the approach for finding $EI_{S_G}(\mathsf{K}, \alpha)$ is terminating and $EI(\mathsf{K}, \alpha) \subseteq EI_{S_G}(\mathsf{K}, \alpha)$.*

Proposition 2 implies that we can use S_G to locate a superset of the affected individuals (defined below).

Definition 17. (Overestimate of Affected Individuals): Let K be a $\mathcal{SHI}$ KB, Q a conjunctive query, S_G the summary root graph for K, α an ABox update and $AI_P(\mathsf{K}, \alpha)$ the set of individuals satisfying conditions 2 or 3 from Definition 14 s.t. $b \in EI_{S_G}(\mathsf{K}, \alpha)$ is used rather than $b \in EI(\mathsf{K}, \alpha)$. Define the overestimate of affected individuals, denoted $AI_O(\mathsf{K}, \alpha)$, as $AI_O(\mathsf{K}, \alpha) = EI_{S_G}(\mathsf{K}, \alpha) \cup AI_P(\mathsf{K}, \alpha)$.

Discussion. It is clear that there are possible limitations to the current approach for determining the overestimate of individuals affected by updates. In particular, if the approach produces an overestimate that is too large, the value of the approach may degrade (note that in the worse case, the number of individuals one would have to check is the same as in the non-incremental case). However, our initial results indicate that the approach is extremely effective. An additional limitation of the approach is the memory overhead imposed by maintaining the summary root graph, which is clearly a trade-off in the approach. One last issue is related to the application of expansion rules on the summary root graph *with respect to the update*. We point out here that in the worst case this could impose overhead that is not practical for the performance demands of some syndication applications; however, our initial results demonstrate that this is not the case. This is because the expansion rules are applied with respect to only the update and not the entire KB. This is actually quite intuitive, as one would expect for updates to only affect a small portion of the KB. Further, if we consider syndication of news feeds for example, one would expect updates to be generally focused on a small number of individuals. This is clearly evident in the financial domain; for example, the Dow Jones Newswires disseminates on average 10,000 news feeds per day[2], which are typically terse and focused on specific companies, industries, etc.

4.3.2 Query Impact on Candidate Individuals

When a query contains roles and more complex query patterns, considering only directly affected individuals as potential new bindings (resp. invalidated bindings for deletions) will not suffice. For example, consider the following query for all companies that have sell recommendations: $\langle x,y \rangle \leftarrow \text{onRecommendation}(x,y) \wedge \text{Company}(x) \wedge \text{SellList}(y)$. Also assume that there is an ABox addition that `Ford` is a *Company* and that after the update, $AI_O(\mathsf{K}, \alpha)$ only includes `Ford`. It is clear we cannot simply consider `Ford` as the only candidate binding for the variables in the query, as there could exist any number of individuals (i.e., instances of *SellList*) related to `Ford` by an *onRecommendation* role. Therefore, it can be observed that the query structure/shape impacts the affected individuals that must be considered as bindings for distinguished variables; we refer to this as the *query impact*.

We now introduce a technique for determining the query impact on the affected individuals, which is a straightforward approach that induces very little overhead. The key insight is that for a new query binding to be entailed (or invalidated in the case of deletions) under syntactic ABox updates in $\mathcal{SHI}$, at least one named individual that is bound to some $x \in V(Q)$ must be in $AI_O(\mathsf{K}, \alpha)$. This, along with the facts that the query is assumed to be connected and only simple roles are used in queries, implies that given an addition update, the query impact on the original affected individuals can be taken into account by also considering all named individuals in the updated completion graph (discussed in Section 4.2) that are reachable from some $x \in AI_O(\mathsf{K}, \alpha)$ by at most n edge traversals (with the direction ignored), where n is the longest path in the query graph. Given this, under additions only the various combinations of individuals in this expanded set of affected individuals need to be considered as possible new bindings for distinguished variables. A similar approach can be used to take into account the query impact under deletions, however the original completion graph (prior to its update) must be used for the search of depth n (i.e., deletions can remove structures from the completion). In the case of deletions, one then needs to recheck any previous binding that contains some individual in the expanded set of affected individuals. Denote by $\mathit{query\_impact}(AI_O(\mathsf{K}, \alpha), Q)$ the extended set of affected individuals under this approach for taking into account the query impact.

It is clear that the previous technique does not leverage the actual structure of the query (i.e., concepts and roles in the query); therefore, we now introduce a more effective approach that exploits such information, but also introduces additional overhead. In this discussion, the approach is only presented for additions, as in typical syndication systems, updates are much more frequently additions; further, extending the approach to deletions is straightforward. We first point out that it has previously been shown that a conjunctive query can be answered be syntactically *mapping* the query into all completion graphs for the KB [20]. More specifically for the DL $\mathcal{SHIQ}$ (also applicable to $\mathcal{SHI}$), [20] shows that given a completion graph G and a query Q, the query can be mapped into G, denoted $Q \hookrightarrow G$. If the query can be mapped into all completions, then the KB satisfies the query. It can be seen that such a mapping is usable when of taking into account the query impact under additions. We note, however, that such a mapping introduces overhead, as it requires that the KB must be extended with $\top \sqsubseteq C \sqcup \neg C$ for each concept $C \in Con(Q)$. In order to further reduce the new candidate bindings, each individual in $AI_O(\mathsf{K}, \alpha)$ can be iteratively substituted into variables in the query; neighbor nodes in the updated completion graph can then be inspected to see if they can be mapped into the remaining nodes (via roles whose labels match the query graph) in the query graph (note that distinguished variables in the query graph are mapped into nodes corresponding to named individuals). If there does not exist a mapping in which a given named individual can be mapped into a particular distinguished variable, then this individual does not need to be considered in the candidate distinguished variable set for this variable; this is because we have just found a completion graph (i.e., model) in which the query cannot be mapped [20]. However, if a named individual can be mapped into a distinguished variable, then we must consider this individual as a candidate binding.

Definition 18. (Query Impact on Candidate Bindings): Let K be a $\mathcal{SHI}$ KB, Q a conjunctive query, α an ABox addition and $G \in Comp(\mathsf{K} \oplus \alpha)$. Define the set of candidate bindings for distinguished variable x under query impact, denoted $A_{QI}(x)$, as follows:

$$A_{QI}(x) = \{a \mid a \in AI_O(\mathsf{K}, \alpha) \wedge Q \hookrightarrow_{\{x \leftarrow a\}} G\} \cup \{a \mid b \in AI_O(\mathsf{K}, \alpha) \wedge Q \hookrightarrow_{\{x \leftarrow a, y \leftarrow b\}} G\}$$

where $Q \hookrightarrow_{\{x \leftarrow a\}} G$ denotes a mapping of Q into G with the restriction that the distinguished variable x must be mapped into the named individual a in the completion graph.

4.3.3 Continuous Query Answering Algorithm

We now describe the algorithm for answering continuous ABox queries. Similar to the discussion presented above, the algorithm is presented in terms of a single query. Note that the algorithm utilizes a combination of both techniques for taking into account *query impact*. It is assumed the KB is first preprocessed such that for each $C \in Con(Q_c)$, an axiom $\top \sqsubseteq C \sqcup \neg C$ is added to the KB; this is necessary only in the cached completion graph for consistency checking and not in the summary root graph. Additionally, it is assumed the summary root graph is created at startup and that the initial set of answers for Q_c is previously determined.

[2]Source: http://www.djnewswires.com/us/djtotalcoverageinfo.htm

Algorithm 1 presents the main continuous query answering algorithm. The approach first locates the affected individuals; this is denoted by *localize_effects*(S_G, α) and takes as input an initial summary root graph S_G and update α and is assumed to both update S_G and return the affected individuals. Additionally, the extended set of affected individuals is found using the simple query impact. If the update is an addition, the set of candidate distinguished variable bindings (determined using Definition 18) is iterated over and checked for entailment. It is assumed that standard techniques for query answering are used (e.g., see [14]). If the update is a deletion, each tuple in the previous answer set is iterated over. Previous tuples that do not contain some individual in *query_impact*$(AI_O(\mathsf{K}, \alpha), Q)$ are still valid, as the update did not affect any of the bound individuals; otherwise, the tuples are rechecked for entailment.

PROPOSITION 3. *Algorithm 1 is sound, complete, and terminating.*

Algorithm 1 *update_query_results*$(\mathsf{K}, S_G, Q_c, R, \alpha)$

Input:
K: initial KB
S_G: summary root graph for K
Q_c: continuous conjunctive query
R: set of all current bindings (answer set)
α: ABox update
Output:
K: updated KB
S_G: updated summary root graph
R: updated bindings (answer set)

1: $\mathsf{K} \leftarrow \mathsf{K} \oplus \alpha$
2: **if** K is not consistent **then**
3: $\quad \mathsf{K} \leftarrow$ Retract α from K
4: $\quad$ **return** K, S_G, R
5: **end if**
6: $AI_O(\mathsf{K}, \alpha) \leftarrow$ *localize_effects*(S_G, α)
7: $QI_S \leftarrow$ *query_impact*$(AI_O(\mathsf{K}, \alpha), Q)$
8: **if** α is an addition **then**
9: $\quad$ **for all** $a_1 \in A_{QI}(x_1), ..., a_n \in A_{QI}(x_n)$ s.t. $x \in DV(Q_c)$ **do**
10: $\quad\quad$ **if** $\mathsf{K} \models Q_c[x_1/a_1, ..., x_n/a_n]$ **then**
11: $\quad\quad\quad R \leftarrow R \cup \{\langle a_1, ..., a_n \rangle\}$
12: $\quad\quad$ **end if**
13: $\quad$ **end for**
14: **else if** α is an deletion **then**
15: $\quad$ **for all** $\langle a_1, ..., a_n \rangle \in R$ **do**
16: $\quad\quad$ **if** $QI_S \cap \{a_1, ..., a_n\} = \emptyset$ **then**
17: $\quad\quad\quad$ **continue**
18: $\quad\quad$ **else if** $\mathsf{K} \not\models Q_c[x_1/a_1, ..., x_n/a_n]$ **then**
19: $\quad\quad\quad R \leftarrow R \setminus \{\langle a_1, ..., a_n \rangle\}$
20: $\quad\quad$ **end if**
21: $\quad$ **end for**
22: **end if**
23: **return** K, S_G, R

5. EMPIRICAL RESULTS

We have implemented the basic functionality of the framework defined in Section 3 and the algorithm presented in Section 4. In the current implementation, publishers can register and publish information (currently all information remains indefinitely valid). Additionally, subscribers can register subscriptions in the form of continuous conjunctive queries that can remain valid for varying amounts of time. In the evaluation, we have focused on information matches as the technical contributions of this work mainly address scalability issues with respect to this problem. Further evaluation of publication matches is left as future work.

We have performed an emperical evaluation using the Lehigh University Benchmark (LUBM) [8] ($\mathcal{SHI}$ expressivity), as it provides a large ABox, therefore simulating a broker with large numbers of persistent publications; additionally, it supplies queries with similar complexity as those used in the examples throughout this paper. It should be noted that 8 OWL equivalent class axioms were changed to subclass axioms, so that the KB was unfoldable. Three queries from LUBM were selected as sample subscriptions and continuously run over a dataset comprised of one university, containing 16,283 individuals and 78,094 assertions. The following three queries were used in the evaluation (LUBM queries 1, 3, and 13 respectively):

$\langle x \rangle \leftarrow$GraduateStudent($x$)$\wedge$takesCourse($x$,`GraduateCourse0`)
$\langle x \rangle \leftarrow$Publication($x$)$\wedge$publicationAuthor($x$,`AssistantProfessor0`)
$\langle x \rangle \leftarrow$Person($x$)$\wedge$hasAlumnus(`University0`,$x$)

In the evaluation, the queries were run over the KB, which was updated with a collection of ABox assertions, simulating newly published information; updates were randomly selected individual type (atomic) and/or role assertions, as this aligns with the types of updates one would expect in syndication systems. Each published document was indefinitely valid. To ensure that some of the updates affected the query results (i.e., subscriptions), there was a 50-percent probability that the update referred to one of the individuals bound to a distinguished variable; therefore, approximately half of the selected updates actually affected the subscriptions. Tests were performed for each update type (additions and deletions) using varying update sizes; namely 1, 5, 10, 15, and 25 assertions. Each test was performed 25 times, and the results were averaged. Lastly, the experiments were performed using a 1.5 GHz processor with 1GB of memory.

In the evaluations, two versions of the DL reasoner Pellet [3] were used; a regular version of the reasoner and an optimized reasoner that used the techniques presented in this paper. In the regular version of the reasoner, the standard query answering algorithm was performed after each update. Additionally, the KAON2[4] OWL-DL reasoner was used in the evaluation. KAON2 reduces OWL KBs to disjunctive datalog and is optimized for query answering. Additionally, KAON2 was used as it provides functionality to add and remove assertions and re-run queries after a KB has been updated (we do note that it is unclear whether KAON2 currently performs view maintenance). Using this as a comparison, we aimed to provide interesting insights into tableau-based algorithms for syndication purposes when compared to other possible approaches.

Results for continuous query answering for the various LUBM queries are presented in Figure 1. Note that the optimized version of Pellet is denoted as "Pellet-C", and the "0" update size value represents the time to run the initial query prior to performing an update (this includes the start-up cost for the continuous query answering approach). In all three queries, the initial query answering times (prior to any update) in both the regular version of Pellet and the optimized version were comparable. This is because the generation of the summary root graph introduced little overhead; specifically, the average time to build the summary root graph was only 2.7 seconds (on average 500 milliseconds larger than the initial consistency check). We note that there is little overhead because of the small amount non-determinism in LUBM (primarily due to making the KB unfoldable). For exposition, however, we investigated the time to build the summary root graph for the original ver-

[3]Pellet project homepage: http://www.mindswap.org/2003/pellet/
[4]KAON2 project homepage: http://kaon2.semanticweb.org/

sion of LUBM, which induce a large amount of non-determinism. It was observed that the average time to build the summary root graph took approximately 26.1 seconds. This demonstrates the expected impact of a substantial amount of non-determinism on the approach; while this introduces overhead, we argue that this is acceptable, as it is only performed once at startup. Further, the generation of the summary root graph was far more efficient than the alternative of saturating the initial KB, as this would not terminate.

For both update types, approximately one to three orders of magnitude performance improvements are achieved over the regular version of Pellet by using the optimized matching algorithm as updates are received. This is due to the incremental consistency checking approach and the reduction of candidate variable bindings. In the evaluation we observed that in all queries, the average incremental consistency checking time was approximately 7 milliseconds, where the normal consistency checking time in Pellet was approximately 2,200 milliseconds. This illustrates the utility of the incremental consistency checking approach for the purpose of syndication. Additionally, in the three queries the actual query answering time (excluding consistency checking) for the regular version of Pellet was on average between 500 to 1,000 milliseconds. In contrast, using the optimizations presented in this work, the average query answering time (excluding consistency checking) was approximately 33 milliseconds; this demonstrates the effectiveness in the reduction of in the number of candidate variable bindings.

With regard to the individual techniques, the evaluation demonstrated the following: given an update of size 1, the average time to apply the completion rules to the updated summary root graph and localize the affected individuals took approximately 0.23 milliseconds. This clearly confirmed our hypothesis that the expansion rules would be applied to only a very small portion of the summary root graph. Even more promising was that the number of affected individuals was proportional to the number of individuals referenced in the update; in particular, for updates of size 1, on average, only 11.144 individuals are affected, amounting to only .068-percent of the entire KB (for increased update size, the number of affected individuals scaled proportionally to the number of individuals referenced in the update). This demonstrates a dramatic reduction in the number of individuals that needed to be considered after each update and shows that the overestimation approach may be usable in practice. Additionally, it can be seen that the optimized version of Pellet outperforms, or performs as well as, KAON2 in all cases. Even more promising is that in query 13, Pellet outperforms KAON2 by almost an order of magnitude.

6. DISCUSSION AND FUTURE WORK

Our preliminary results demonstrate that the matching approach presented in this paper can scale to a few hundred subscriptions under publish frequencies similar to that of the Dow Jones Newswire (i.e., 10,000 per day ~ approximately 7 per minute). While this may be an acceptable workload for a wide range of syndication applications (e.g., filtering financial news feeds within small to medium investment banks), for larger scale applications, additional research is necessary. One direction that we plan to investigate is leveraging the overlap and/or subsumption between registered subscriptions. Additionally, we plan to investigate distributed OWL-based syndication frameworks (i.e., more than one broker), as this will provide increased scalability.

In this paper, we have primarily addressed providing a more practical approach for finding information matches in OWL-based syndication systems. As mentioned earlier, there has been extensive work on finding minimal justifications in OWL KBs [15]. Using such an approach, it is easy to extend information matches to publication matches. Further, initial results presented in [15] for finding justifications demonstrates that such an approach may be practical. In future work, we will explore the usage of these techniques for extending our current work.

We also feel that there is substantial room extending the current reasoning approaches. This includes developing additional optimizations for reasoning through changing KBs, as well as extending the current techniques to a larger portion of OWL; in particular, we feel it is certainly possible to lift the restriction that the KB be unfoldable, and we will address this in future work. Additionally, while our initial results demonstrate that the overhead of the advanced form of query impact is acceptable, we plan to further investigate the tradeoffs between the two variants presented here.

Lastly, in real world domains, it is often the case that conflicting information is disseminated. Depending on the ontologies used within such a syndication framework, this could lead to inconsistencies. Currently, we are working on developing revision techniques for OWL-DL KBs and hope to apply such efforts to resolving inconsistencies encountered in syndication systems [11].

7. RELATED WORK

There has been substantial research on syndication systems, with a transition to more expressive approaches for representing subscription requests and published information. These have included keyword-based approaches (e.g., [19]), attribute-value pairs (e.g., [1]), XML (e.g., [4]) and recently RDF-based approaches (e.g., [23]). The approach presented here provides increased expressivity for representing published information (i.e., complex logic descriptions of published content can be defined that are not representable using previous techniques).

[22, 17] proposes a DL-based approach for syndication in which DL concepts are used for both subscription requests as well as published documents/data. [9] presents a DL-based syndication approach in which the subscriber registers queries (restricted to single, named concepts) that model their interests and published data is modeled as ABox assertions. [9] also presents two optimizations. Frst, the authors propose inducing a partial ordering upon all registered queries by their subsumption relations; more general queries are answered first, thereby reducing the number of individuals that must be considered for more specific queries. [9] also proposes disregarding previous individuals that satisfied registered queries when data is published. Our approach differs as we support complex conjunctive queries, allow subscription and published document expiration times, etc. Additionally, we address the performance bottlenecks of DL-based syndication further.

There has been substantial work in continuous query answering in relational databases and datalog (e.g., [21, 5]). While related, the work presented here addresses a more expressive formalism.

8. CONCLUSION

In this paper, we have formalized a OWL-based syndication framework in which DL reasoning is the primary means for matching newly published information with subscription requests. We then addressed one of the main limitations with such a syndication framework, namely efficiently matching new information with registered subscriptions; to this end, we formally defined continuous queries (i.e., subscriptions) for DL KBs and presented a novel algorithm for continuous query answering. Lastly, an evaluation of the query answering approach for syndication purposes has been presented, demonstrating dramatic performance improvements.

We would like to thank Jennifer Golbeck, Yarden Katz, Vladimir Kolovski, Bijan Parsia, Evren Sirin, and Taowei Wang for all of

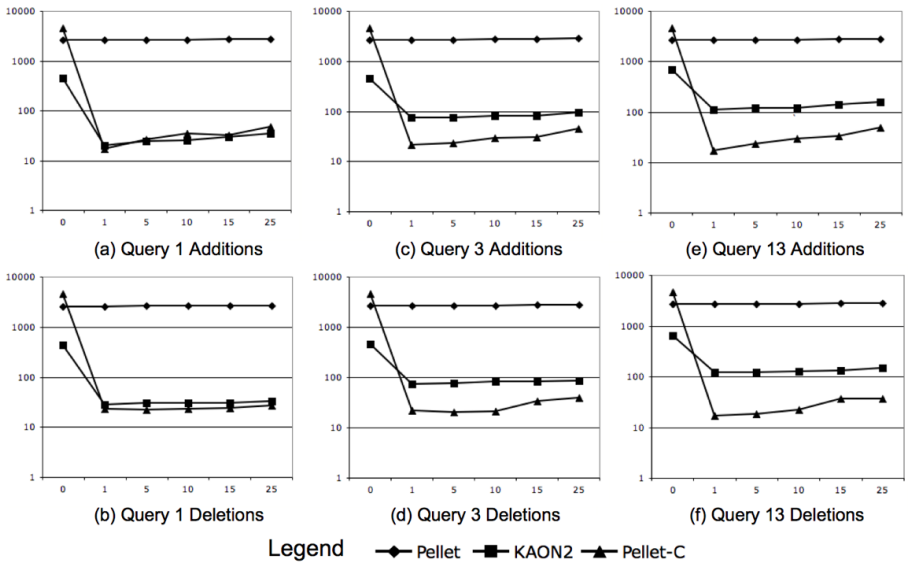

Figure 1: Continuous query answering for LUBM queries. Times (log.) in milliseconds along Y-axis. Update size along X-axis.

their contributions to this work. This work was supported by grants from Fujitsu, Lockheed Martin, NTT Corp., Kevric Corp., SAIC, the National Science Foundation, the National Geospatial-Intelligence Agency, DARPA, US Army Research Laboratory, and NIST.

9. REFERENCES

[1] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra. Matching events in a content-based subscription system. In *Symposium on Principles of Distributed Computing*, 1999.

[2] F. Baader and B. Hollunder. Embedding defaults into terminological representation systems. *Journal of Automated Reasoning*, 14:149–180, 1995.

[3] F. Baader and W. Nutt. Basic description logics. In *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 43–95. 2003.

[4] Y. Diao, S. Rizvi, and M. Franklin. Towards an internet-scale xml dissemination service. In *Proc. of Int. Conf. on Very Large Data Bases*, 2004.

[5] G. Dong and R. W. Topor. Incremental evaluation of datalog queries. In *Proc. of Int. Conf. on Database Theory*, 1992.

[6] G. Flouris, D. Plexousakis, and G. Antoniou. On applying the agm theory to dls and owl. In *Proc. of Int. Semantic Web Conf.*, 2005.

[7] B. Glimm, I. Horrocks, C. Lutz, and U. Sattler. Conjunctive query answering for the description logic $\mathcal{SHIQ}$. In *Proc. of Int. Joint Conf. on Artificial Intelligence*, 2007.

[8] Y. Guo, Z. Pan, and J. Heflin. Lubm: A benchmark for owl knowledge base systems. *Journal of Web Semantics*, 3(2):158–182, 2005.

[9] V. Haarslev and R. Möller. Incremental query answering for implementing document retrieval services. In *Proc. of Int. Workshop on Description Logics*, 2003.

[10] C. Halaschek-Wiener and J. Hendler. Expressive logic-based syndication on the web. In *UMIACS Technical Report*. http://www.mindswap.org/~chris/publications/Syndication-OWL-TR2006.pdf.

[11] C. Halaschek-Wiener, Y. Katz, and B. Parsia. Belief base revision for expressive description logics. In *Proc. of Workshop on OWL Experiences and Directions*, 2006.

[12] C. Halaschek-Wiener, B. Parsia, and E. Sirin. Description logic reasoning with syntactic updates. In *Proc. of Int. Conf. on Ontologies, Databases, and App. of Semantics*, 2006.

[13] I. Horrocks and U. Sattler. A tableaux decision procedure for SHOIQ. In *Proc. of Int. Joint Conf. on Artificial Intelligence*, 2005.

[14] I. Horrocks and S. Tessaris. Querying the semantic web: a formal approach. In *Proc. of Int. Semantic Web Conf.*, 2002.

[15] A. Kalyanpur. Debugging and repair of owl ontologies. In *Ph.D. Dissertation, University of Maryland, College Park*, 2006.

[16] L. Lakshmanan and S. Parthasarathy. On efficient matching of streaming xml documents and queries. In *Proc. of Int. Conf. on Extending Database Technology*, 2002.

[17] L. Li and I. Horrocks. A software framework for matchmaking based on semantic web technology. In *Proc. of Int. World Wide Web Conf.*, 2003.

[18] H. Liu, C. Lutz, M. Milicic, and F. Wolter. Updating description logic aboxes. In *Int. Conf. of Principles of Knowledge Representation and Reasoning*, 2006.

[19] B. Oki, M. Pfluegl, and D. Skeen. The information bus: An architecture for extensible distributed systems. In *Proc. of Symposium on Operating Systems Principles*, 1993.

[20] M. Ortiz, D. Calvanese, and T. Eiter. Characterizing data complexity for conjunctive query answering in expressive description logics. In *Proc. of Nat. Conf. on Artificial Intelligence*, 2006.

[21] D. B. Terry, D. Goldberg, D. Nichols, and B. M. Oki. Continuous queries over append-only databases. In *Proc. of Int. Conf. on Management of Data*, 1992.

[22] M. Uschold, P. Clark, F. Dickey, C. Fung, S. Smith, S. U. M. Wilke, S. Bechhofer, and I. Horrocks. A semantic infosphere. In *Proc. of Int. Semantic Web Conf.*, 2003.

[23] J. Wang, B. Jin, and J. Li. An ontology-based publish/subscribe system. In *Proc. of Int. Conf. on Middleware*, 2004.