Modeling Associations through Intensional Attributes

Andrea Presa, Yannis Velegrakis, Flavio Rizzolo, and Siarhei Bykau

University of Trento
{apresa,velgias,flavio,bykau}@disi.unitn.eu

Abstract. Attributes, a.k.a. slots or properties, are the main mechanism used to define associations between concepts or individuals modeling real world entities in a knowledge base. Traditionally, an attribute is defined by an explicit statement that specifies the name of the attribute and the entities it associates. This has three main limitations: (i) it is not easy to apply to large amounts of data, even if they share the same characteristics, since explicit definitions are needed for each concept or individual; (ii) it cannot handle future data, i.e., when new concepts or individuals are inserted in the knowledge base their attributes need to be explicitly defined; and (iii) it assumes that the data engineer, or the user that is introducing a new attribute, has access and privileges to modify the respective objects. The above may not be practical in many real ontology application scenarios. We are introducing a new form of attribute in which the domain and range are not specified explicitly but intensionally, through a query that defines the set of concepts or individuals being associated. We provide the formal semantics of this new form of attribute, describe how to overcome syntax constraints that prevent the use of the proposed attribute, study its behavior, show efficient ways of implementation, and experiment with alternative evaluation strategies.

1 Introduction

We are witnessing a tremendous increase in the amount of data that is becoming available online. To effectively access this data, we need to be able to successfully understand its semantics. Schemas have to a large degree contributed towards that direction, but they have not fully fulfilled their role – they are mainly driven by performance or technical motivations and do not always communicate accurately the semantics of the data. For modeling complex data semantics, ontologies, rather than schemas, are typically used. Ontologies are free from the structural restrictions that schemas have. A ten thousand feet view of an ontology is a collection of concepts (or classes) and individuals (or instances) associated through *isA* and *attribute* relationships. In the ontology jargon [1], the latter are referred to as *slots* or *properties* and they are used to describe features of a class or an individual. Each attribute has a type and can be restricted to draw its values from a specific pool of values.

A limitation of the attribute modeling constructs in current ontology formalisms is their static nature. More specifically, the existence of an attribute between two concepts or individuals depends solely on whether the slot has been explicitly defined or not. This prevents the implementation of batch assignment of attributes to groups of concepts/individuals that are currently present in the knowledge base or that may appear in the future. For instance, in many practical scenarios, attributes are assigned to individuals based on some common characteristics. Currently, this task requires first finding

the individuals that have these characteristics, iterate over them, and explicitly assign to them the attribute of interest. Furthermore, if one or more individuals satisfying these characteristics are introduced at some future point in time, they will not be automatically assigned the attribute, unless a special ad-hoc mechanism has been put in place, or the ontology administrator manually assigns it to each such individual.

A different issue related to the current ontology mechanisms has to do with the way additional/super-imposed information can be attached to the structures of a knowledge base. Recall that ontologies is one of the main vehicles of communicating data semantics. To better achieve that goal, designers typically attach to the ontology constructs additional information that is not considered part of the ontology itself, yet assists in better communicating the data meaning. The RDF/RDFS standard [2, 3] has provisioned a special single-string text field named *comment* for that purpose. The *comment* mechanism has two main limitations. First, it confines the ontology engineer to provide a single piece of plain text, whereas recording a comment that has some structure is typically more useful. For instance, we may want to insert a comment on an RDF resource along with the date and the name of the person that created the comment. Current practices include all that information in the comment text, but the text needs to be parsed every time the individual parts are to be identified. Second, attaching information to existing concepts or individuals of an ontology means that the user needs to have the privileges to do so. This is not always the case since many different users, other than the ontology owner, may need to add information of different kinds.

In this work, we advocate the need for *intensional attributes*[1], i.e., attributes whose domain and range have been intensionally defined. Individuals are assigned to the intensional attributes' domain and range in a similar fashion in which they are assigned to the extensions of defined concepts in Description Logics (DL) TBoxes [4] (as opposed to the explicit way individuals are assigned to the primitive concepts). To some extent, this kind of definition looks also similar to the way derived elements in UML [2] are defined. However, the notion of intensional attributes is fundamentally different from both derived elements and derived concepts. Derived elements or concepts are used to describe entities, while intenional attributes are used to describe derived relationships between entities. In that sense, intensional attributes do not replace but actually complement DL TBoxes and UML derived elements. In our solution we employ queries in order to specify the domain and range of the intensional attributes. We claim that queries are an excellent tool to implement intensional attributes since they provide the ideal mean to refer to sets of data declaratively. The idea of using queries for intensional definitions has also been proposed in other forms in different fields [5–9]. However, to the best of our knowledge, this is the first effort towards using that idea for attributes in ontologies to tackle the previously presented issues.

Our contributions are the following: (i) we redefine the notion of an attribute to include those for which the associated concepts or individuals are described intensionally; (ii) we describe how we can overcome RDF/RDFS limitations that prevent the use of the proposed kind of attributes; (iii) we describe how the new form of association can be realized in OWL and RDF ontologies; (iv) we provide techniques to efficiently

[1] the term *intensional* should not be confused to the term *intentional*.

[2] http://www.uml.org

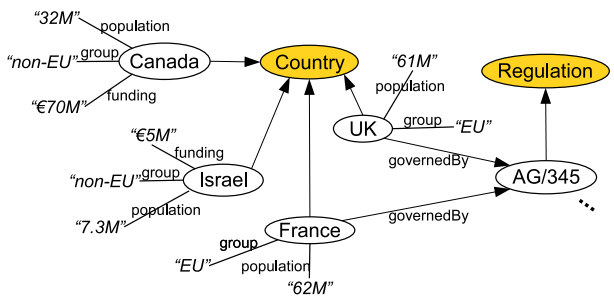


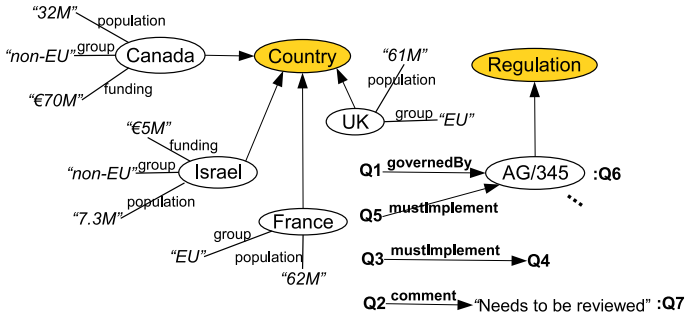
Fig. 1. Ontology example

implement ontology browsing and query answering for this new form of association; (v) we experimentally evaluate alternative techniques and describe our findings.

2 Illustrative Example

To motivate our proposal and illustrate our solution, we describe here an example drawn from a real application with which we have worked. Consider a financial department that handles projects funded by the European Union (EU). The EU not only funds projects in countries that are members of the EU block but also in certain countries outside it. However, the funding is governed by different regulations depending on whether the recipient country is inside or outside the block. Consider an ontology modeling countries as illustrated in Figure 1. Countries (class Country) are distinguished as EU or non-EU through the attribute group. Some readers may (rightfully) claim that the right modeling of this situation is through two sub-classes of the class Country; however, this was modeled in practice by an attribute. Each EU country is associated through the governedBy attribute to the AG/345 regulation (an instance of the Regulation class). Each such attribute has been explicitly introduced to the respective countries by the ontology engineer after checking whether the country belongs or not to the EU block. This is a laborious task; it requires the data administrator to manually visit each country's data, test whether it belongs to the EU block and assign the specific attribute. It also requires continuous monitoring, so that if a new EU country is introduced, the attribute governedBy with value AG/345 will be assigned to it. Furthermore, it may be the case that the specific data engineer does not have full privileges to modify the individual country.

Our proposal is that the administrator could introduce instead an intensional attribute governedBy as illustrated in Figure 2. The attribute has at one end (as a range) the individual AG/345, and on the other (as a domain) the query Q1 that selects all the individual countries belonging to the EU block. For ease of presentation, we use a simplified notation of queries, i.e., $x.a{=}v$ indicates that the attribute a of class/individual x has a value v. In reality, we are using an actual query language, but the details will be described later on. Note how we use the attribute rdf:type=C in order to indicate that the results of a query are individuals of some specific class C, and that in order to mention a specific individual or class we use a trivial form of a query (e.g., query Q6 in Figure 2).



Q1: { x such that x.rdf:type=Country and x.group="EU" }
Q2: { x such that x.rdf:type=Country and x.population $\leq$ 20) }
Q3: { x such that x.rdf:type=Country }
Q4: { x such that x.rdf:type=Regulation and x.code="EMR*" }
Q5: { x such that x.rdf:type=Country and x.funding $>$ 10M and x.funding $<$ 100M }
Q6: { x such that x=AG/345 }
Q7: { x such that x="Needs to be reviewed" }

Fig. 2. Ontology example with intensional attributes

Apart from the obvious saving in space and human effort for not having to repeat the attribute for every EU country, using the intensional approach has also the additional advantage of covering future data. In particular, if a new country becomes a member of the EU, an ontology administrator following the traditional approach would have to explicitly associate it to the regulation AG/345. In contrast, using our proposed approach, the administrator has to do no action: the moment the country becomes a member of the EU (i.e., by setting the group attribute to the EU value) it automatically satisfies the conditions of query Q1 and becomes part of its answer set, which has the effect of attaching attribute governedBy with value AG/345 to it.

As a different example, assume that a user would like to add some super-imposed information on the countries, indicating that every country with a population smaller than 20 millions will have to be reviewed. To add this kind of "annotation" on the countries, the user will have to explicitly introduce it either by utilizing the *comment* feature provided by the RDF/RDFS model (assuming of course that the ontology is expressed in RDF/RDFS) or by adding a special attribute with the appropriate text to each such country. Allowing the user to add attributes of this kind, as suggested by the latter solution, may not always be feasible or desirable. It may not be feasible because the user may not have permissions to edit the ontology. Even if this is not the case, it may not be desirable since adding attributes to the ontology concepts and individuals without some control mechanism may alter their semantics. In contrast, by using an intensional attribute between a string with the aforementioned statement and the query Q2 shown in Figure 2, the desired result can be achieved even without having permissions to modify the Country individuals.

Note that queries may exist on either or both parts of an intensional attributes. An example demonstrating such a situation is the following. EU has introduced a set of financial regulations, containing the code "EMR", that need to be followed by every coun-

try that is to receive EU funding. Without intensional attributes, the ontology needs to have on every country a number of attributes mustImplement one for each regulation that contains the encoding "EMR". However, using the intensional features, just the attribute mustImplement with domain Q3 and range Q4 as illustrated in Figure 2 will be enough. (Query Q3 returns the set of all countries and Q4 returns all regulations whose code attributes contain the string "EMR".)

Intensional attributes may share the same name as long as their domain and range queries are different. For instance, consider now that all countries, EU and non-EU, have to implement regulation AG/345 if the funding they receive is between 10 and 100 million (illustrated by attribute mustImplement with Q5 as domain in Figure 2). Since all answers of Q5 are countries, then Q5 is included in Q3 and thus all members of Q5 have already a mustImplement attribute defined (with values from the regulation individuals in the answer set of query Q4). However, the definition of the new intensional attribute will include AG/345 as an additional regulation that only members of Q5 must have. This result could *not* have been achieved by only one intensional attribute mustImplement with domain Q5∪Q3 and range Q4∪Q6 because this definition would add AG/345 to all countries, even to those whose funding is outside the range specified by Q5 (i.e., outside 10M<funding<100M). Thus, in order to correctly capture the requirement, a new intensional attribute mustImplement from Q5 to Q6 has to be present.

3 Intensional Attributes

As a knowledge base we follow the traditional definition that consists of a set of classes, individuals and attributes. In particular, we define a *knowledge base* S as the set of *classes* $\mathcal{C}$, *individuals* $\mathcal{I}$ that are instances of classes in $\mathcal{C}$, *names* $\mathcal{N}$ and *literals* $\mathcal{L}$ belonging to atomic types in a set $\mathcal{T}$. A knowledge base also contains a set of *attributes* $\mathcal{A}$. An attribute is a named association between a class and a type or another class, or between an individual and an atomic value (i.e., literal) or another individual. In other words, $\mathcal{A} \subseteq (\mathcal{C} \times \mathcal{N} \times (\mathcal{T} \cup \mathcal{C})) \cup (\mathcal{I} \times \mathcal{N} \times (\mathcal{A} \cup \mathcal{I}))$. According to the above definition, an attribute can be represented as a triple $\langle s, p, v \rangle$, where s is a class or an individual, p is a name (typically referred to as the name of the attribute), and v is a class, a type, an individual or a literal value.

We extend the above traditional definition of a knowledge base to include a set of *intensional attributes*. An intensional attribute is defined as a triple $\langle q_d, n, q_r \rangle$, where q_d is a query that returns a set of classes or individuals, and q_r a query that returns a set of classes, individuals, or literals. A knowledge base with intensional attributes will be referred to as an *intensional base*.

Example 1. Figure 2 shows an example of an intensional base. Triples ⟨Q3,mustImplement,Q4⟩ and ⟨Q5,mustImplement,Q6⟩ are examples of intensional attributes. In particular, ⟨Q1,governedBy,Q6⟩ and ⟨Q2,comment,Q7⟩ are attributes in which one of the two queries returns always only one element in its answer set: Q6 returns always the individual AG/345, and Q7 the literal "Needs to be reviewed". ∎

Intuitively, an intensional attribute $\langle q_d, n, q_r \rangle$ is a short-hand for a set of attributes (each one with the same name n) between a member in the answer set of query q_d and a member in the answer set of the query q_r.

Example 2. Consider again the example of Figure 2. Attribute mustImplement is equivalent to the addition of an attribute mustImplement from every element in the result set of Q3 (i.e., every country) to every element in the result set of query Q4, i.e., every regulation containing the string "EMR" in its code. Similarly, the semantics of governedBy is to associate to every EU country a governedBy attribute to regulation AG/345. Finally, the semantics of comment is to attach the string literal "Needs to be reviewed" to every country with population smaller than 20 millions. ∎

The formal semantics of an intensional base, and consequently of the intensional attributes in it, are realized through the notion of the *canonical base*. Intuitively, a canonical base is an intensional base in which every intensional attribute has been replaced by the set of traditional attributes it represents.

Definition 1. *Let $S{=}\langle \mathcal{C}, \mathcal{I}, \mathcal{T}, \mathcal{L}, \mathcal{A}, \mathcal{D} \rangle$ be an intensional base, where $\mathcal{C}, \mathcal{I}, \mathcal{T}, \mathcal{L}, \mathcal{A}, \mathcal{D}$ are the classes, individuals, types, literals, attributes and intensional attributes respectively. The canonical base of S, denoted as $Can(S)$, is a knowledge base $\langle \mathcal{C}, \mathcal{I}, \mathcal{T}, \mathcal{L}, \mathcal{A}' \rangle$ for which $A'{=}A \cup \{\langle r_d, n, r_r \rangle \mid \exists \langle q_d, n, q_r \rangle \in D$: $r_d \in eval(q_d) \wedge r_r \in eval(q_r)\}$. The function $eval$ is a function that evaluates the query provided as an argument and returns its results.* ∎

The semantics of the knowledge base with intensional attributes are the same as the semantics of its canonical base.

Definition 2. *Let q be a query or a reasoning task over an intensional base S. The result of q over S is defined as the result of the evaluation of q over $Can(S)$.* ∎

Note that according to the above definition, no extension, special adjustments or operators need to be added to the query language in order to allow querying intensional bases. Of course, what needs to be adjusted is the actual evaluation mechanism which will be the topic of Section 5. Furthemore, note that when evaluating queries used in intensional attributes we do not consider other intensional previously defined attributes. That way, we avoid having recursive definitions.

To record intensional attributes in RDF, we have extended the RDF/RDFS model by introducing a new class Query. Every query is represented as an instance of that class, that has an attribute expression which is a string representing the query expression. Furthermore, we have created the class Intensional Attribute as a subclass of Property in which we have restricted the attributes domain and range to instances of the class Query. Figure 3 illustrates these extensions. The figure provides the major RDF/RDFS constructs as described by W3C [3] along with our proposed extensions which are indicated in the figure with shadowed nodes.

The queries used in the intensional attributes are queries supported by the system on which the framework runs. Thus, the complexity of supporting intensional attributes is only restricted by the complexity of the queries supported by the reasoner of the system. No additional reasoning is required (in the worst case scenario) than evaluating

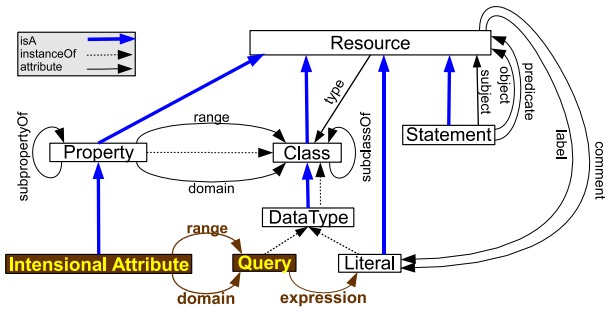


Fig. 3. The RDF/RDFS Schema model, extended to support intensional attributes

the query. However, for the practical class of queries that we consider in the next section and by using the evaluation technique that is described there, it will be shown that the complexity for the reasoner is linear to the size of the conditions in the query expression and the number of the attributes that exist in the knowledge base.

4 Realization of Intensional Attributes

To better understand the intuition behind intensional attributes, one can consider the paradigm of *defined concepts* in Description Logics (or DL) [10]. A DL terminological box (TBox) consists of two kinds of concepts, the *primitive* and the *defined*. Primitive concepts are those whose extensions are specified by explicit statements associating each individual to the respective concept. A defined concept, on the other hand, is a concept whose extension is specified through a logical expression. Every individual that satisfies the expression of a defined concept is considered automatically a member of its extension. Our proposed intensional attributes can be seen as an extension of the idea of defined concepts to attributes.

To realize intensional attributes, a solution that easily comes in mind is to exploit the ability of RDFS on defining domain and range constraints. Unfortunately, we show next that this is not possible. Let $\langle q_d, m, q_r \rangle$ be an intensional attribute. One can introduce two defined concepts C_d and C_r, using the query expressions q_d and q_r, respectively. Then, attribute m can be defined with concept C_d as domain and concept C_r as range. A limitation of this approach is that one needs to introduce two new defined concept for each different intensional attribute. In a relational database, this is similar to create a view for every query that is to be answered in the system. Naturally, this is not practical, first because the number of the queries may be large, and second, because access rights may not permit users to create new views (respectively, concepts). Furthermore, the knowledge base may be based on core DL, as most of the DL systems, which does not support the definition of attributes on defined concepts.

An alternative solution that avoids the introduction of new concepts is the use of the query expression directly in the definition of the property. For instance, the definition of property governedBy in Figure 2, could have been achieved by using in the domain part the OWL abstract syntax expression SubClassOf(intersectionOf(Country

restriction(group value("EU"))) restriction(governedBy value(AG/345))) which represents query $Q1$. This idea, no matter how close it appears to be to what we are trying to achieve, is fundamentally different from the one behind intensional properties. According to the OWL specifications [11], the semantics of the domain and range classes of a property is that all its property instances must be between instances of these classes; however, the property definition itself does not specify what specific individuals participate in the property instance. In other words, defining the property hasCar with class Person as a domain and class Car as a range, does not automatically associate any of the individuals in the extension of Person with any of the individuals in the extension of Cars. In contrast, the semantics of the intensional attributes is that every individual in the extension of the domain class is *automatically* associated through the property with every individual in the extension of the range class.

For expressing the queries used in the definitions of the intensional attributes, we decided to use SPARQL [12] since it is one of the popular ontology query languages. We need to note, however, that the selection of the language is a design choice and does not affect the semantics of the intensional attribues. Since the role of intensional attributes is to associate classes or individuals with other classes, individuals or literals, it is natural to assume that each such query returns a set of one of those three kinds. Using SPARQL syntax, we can express queries Q1 through Q7 from Figure 2 as follows:

Q1: select ?x where {?x rdf:type Country . ?x group "EU"}
Q2: select ?x where {?x rdf:type Country . ?x population ?p . FILTER (?p <=20)}
Q3: select ?x where {?x rdf:type Country}
Q4: select ?x where {?x rdf:type Regulation . ?x code "EMR"}
Q5: select ?x where {?x rdf:type Country . ?x funding ?f . FILTER (?f > 10M && ?f < 100M)}
Q6: select distinct ?x where {?x rdf:type ?y . FILTER (str(?x) = "AG/345")}
Q7: select distinct ?x where {?z ?y ?x . FILTER (?x = "Needs to be reviewed")}

Note that, in contrast to relational query languages, in SPARQL it is not possible to return a constant that does not exist in the knowledge base. For instance, in SQL we can say select "mystring" which will return an answer set containing only the value ''mystring". To achieve the same behavior in SPARQL, we need to have somehow the string explicitly stored in the knowledge base. To overcome this limitation, in our implementation we use a special class that stores as instances all the possible strings used in the queries. Thus, any query that needs to return only one value, like Q7 above, will indeed return the expected result. This is not a limitation of our approach, rather a way to overcome an SPARQL restriction.

5 Supporting Intensional Attributes Functionality

To support the intensional attribute functionality in a way that is transparent to the end user, we have investigated three main approaches that we describe next.

5.1 The Materialized Approach

The idea of the *materialized* approach is that all intensional attributes are materialized as regular attributes. When a new intensional attribute $\langle q_d, m, q_r \rangle$ is introduced in the

knowledge base, the queries q_d and q_r are evaluated generating the result sets S_D and S_R. Then, for every member d in S_D and for every member r in S_R an attribute $\langle d, m, r\rangle$ is introduced in the knowledge base. Therefore, with the materialized approach, finding the intensional attributes of a class or individual is reduced to finding its regular attributes. When a new class/individual is introduced in the knowledge base, the system needs to find whether it needs to be assigned one or more intensional attributes, (recall the ability of the intensional attributes to be assigned to future data). To do this, the system has to go through all the queries of the intensional attributes and evaluate them in order to determine whether the class or individual is part of the answer set of any of them. One possible optimization for this process is to use an indexing mechanism similar to the one described in Section 5.3.

5.2 The Lazy Approach

The other extreme of the materialized approach is the *lazy* approach. According to it, no materialization takes place and the systems keeps only the definitions of the intensional attributes. This offers great space savings compared to the materialized approach. Furthermore, insertion or deletion of data does not require any action.

The limitation of the lazy approach is its high cost during browsing and query answering. The system can always find whether a class or an individual x has an intensional attribute $\langle q_d, m, q_r\rangle$ by simply evaluating queries q_d and q_r and testing whether x belongs to their answer set. Given a class or an individual x, to find its intensional attributes the system has to evaluate all the queries of all the stored intensional attribute definitions. This is the same process that the materialized approach had to perform when some new data is inserted into the intensional base. However, assuming that updates are not performed very often, we rather pay such a cost during data modification than during browsing or query answering.

5.3 The Indexed Approach

To avoid the two extremes of the lazy and the materialized approach, we looked for a method with a reasonable performance during query answering and a reduced cost in terms of space. As a result, we developed the *indexed* approach. Its basic idea is that, instead of fully materializing the intensional attributes as in the materialized approach, we create special index structures that allow us to find the intensional attributes of a given class/individual with a much lower cost than the one required when evaluating all the queries in the lazy approach.

In this approach, we restrict our attention to the special class of queries consisting of a set of attribute conditions. This class is equivalent to the select-project-join queries in relational database systems which have been found to constitute a large portion of those met in real application scenarios [7]. In what follows, we concentrate on select-project queries for reasons of presentation. The addition of the join does not alter the methodology or the results. Thus, we assume the indexed approach is used for queries of the SPARQL form:

$$\text{select } ?o \text{ where } \{?o \text{ rdf:type } c \ . \ Conds\}$$

where c is a specific class (not a variable) and $Conds$ is a series of conditions combined by the "." operator. Each condition in $Conds$ is of the form: $?o$ attributeName

DTable

Qd	Name	Qr
Q1	governedBy	Q6
Q2	comment	Q7
Q3	mustImplement	Q4
Q5	mustImplement	Q6
...	...	...

MTable

Q	Max	Cr
Q1	2	0
Q2	1	0
Q3	1	0
Q4	2	0
Q5	1	0
Q6	1	0
...	...	...

ETable

Attr	Value	Q
group	EU	Q2
hasURI	"AG/345"	Q6
hasURI	"EMR"	Q4
rdf:type	Country	Q1
rdf:type	Country	Q2
rdf:type	Country	Q3
rdf:type	Country	Q5
rdf:type	Regulation	Q4
...	...	...

ITable

Cond	Q
population $\leq$ 00020	Q2
___funding $>$ 00010	Q5
___funding $>$ 00020	Q10
___funding $>$ 00100	Q12
___funding $<$ 00010	Q11
___funding $<$ 00050	Q10
___funding $<$ 00100	Q5
...	...

Fig. 4. DTable, MTable, ETable and ITable examples

$?v$. FILTER($?v\langle OP\rangle attributeValue$). The operator $\langle OP\rangle$ can be one of the $=$, $<$, $\leq$, $>$ or $\geq$. The meaning of such a condition is that the class or individual o has an attribute $attributeName$ whose value is related to the $attributeValue$ as specified by the $\langle OP\rangle$ operator. For readability purposes, we will write conditions like the above as: $attributeName\langle OP\rangle attributeValue$.

The index consists of four tables: DTable, MTable, ETable, and the ITable. (Figure 4 illustrates the contents of the tables for the knowledge attributes in the intensional base of Figure 2.) Note that their structure permits the implementation in both relational and triple store systems. We explain their structures, roles and uses in the next paragraphs.

The DTable is a 3-column table used to record the list of the defined intensional attributes. The first and last columns record the domain and range queries, respectively, while the second one records the attribute names. More specifically, a tuple $[q_d, m, q_r]$ in DTable indicates the existence of an intensional attribute $\langle q_d, m, q_r\rangle$.

The MTable is also a 3-column table. It contains one entry for each query. The first column of the table specifies the query. The second column is an integer and specifies the number of equality conditions, i.e., conditions of the form $attributeName$=$attributeValue$, that the respective query has. The use of the third attribute will be described shortly. We require every query to have at least one equality condition on the type. We explicitly add condition rdf:type=owl:Thing to any query with no type condition. This guarantees at least one entry in the MTable for every query. Note, for instance, that query Q2 has in the middle column the value 1, since the only equality condition is the one on the type (The condition on the population is not an equality).

The ETable is the placeholder of the equality conditions of the queries in the intensional attributes. It consists of three columns recording the attribute name, the value of the equality condition, and the query name, respectively. (We assume that each query used in the intensional base is assigned a unique-name identifier.) All three values are stored as strings. Figure 4 contains an ETable for the queries in our running example.

Tables MTable and ETable need to be updated when intensional attributes are introduced or removed from the system. When an intensional attribute $\langle q_d, m, q_r\rangle$ is introduced, it is first inserted in table DTable, and then its queries q_d and q_r are analyzed. For every equality condition $cond$ they contain, a tuple $[q, cond]$ is inserted in ETable, where q is q_d or q_r. In addition, the tuples $[q_d, n_d, 0]$ and $[q_r, n_r, 0]$ are inserted in table MTable, where n_d and n_r are integers indicating the number of equality conditions of q_d and q_r, respectively. In the case of a deletion of an intensional attribute $\langle q_d, m, q_r\rangle$,

the only action required is the removal from the tables DTable, MTable, and ETable of any tuple referring to query q_d and q_r.

Assume now that the system needs to find the intensional attributes of a class or an individual o. The system has first to find the queries that have o in their answer set. (To simplify the presentation we will ignore for the moment the inequality conditions.) To do so, all the values in the column Cr of MTable are initially set to 0. Then for each non-intensional[3] attribute $attrName$ with value $attrValue$, a lookup is performed on table ETable for tuples containing values *attrName* and *attrValue* in columns Attr and Value, respectively, and the Q column value of those tuples is retrieved. For each query q in the retrieved set, the Cr value of the tuples in table MTable that have q in column Q is increased by one. At the end of the process, the queries for which their MTable has the same value in columns Max and Cr are those for which the non-intensional attributes of o satisfy the query condition, thus o is in their answer set. Let $\mathcal{Q}$ be the set of such queries, and let us call it the *candidate set*. The intensional attributes of o are those in table DTable that have in column Qr or Qd a query that belongs in the candidate set $\mathcal{Q}$.

Example 3. As an example of the described process, consider the index structures illustrated in Figure 4, and the individual Canada of Figure 2. The specific individual has four (attributes, value) pairs: rdf:type=Country, group=non-EU, funding=70M and population=32M. A set of polling requests on ETable, one for each (attribute, value) pair, shows that Q1, Q2, Q3 and Q5 satisfy rdf:type=Country (no other pair is satisfied). Then, we increase by 1 the Cr column of the MTable for the tuples of the four queries mentioned above. The result table MTable will be the one of Figure 4 with the tuples of Q1, Q2, Q3 and Q5 having value 1 in column Cr. Among them only the Q2, Q3 and Q5 agree with the value in their respective column Max, which means that Canada satisfies all equality conditions of Q2, Q3 and Q5, thus, it belongs in their answer set. It does not, however, belong to the answer set of query Q1 since its Cr column value 1 is smaller than its Max column value 2. ∎

In the discussion so far we have considered only equality conditions. We see next how inequality conditions, i.e., conditions involving $>$ and $<$, are handled. Table ITable serves that purpose. It consists of two columns. The first column (Cond) is a column that contains the inequality conditions of every query used in the intensional attributes. The second column specified the query in which this inequality condition exists. If the same condition appears in more than one query, then the table has multiple entries, one for each query in which it appears. The values in the column Cond are strings of a fixed $2N$+1 character length used to record an inequality condition. The first N characters are used for the attribute name, the next character for the operator, and the last N for the value. If the attribute name is smaller than N in length, it is padded with underscores or zeros, depending on whether it is a string or a number. For instance, if N is 15, the string representation of the condition $population < 20$ is "_____population<000000000000020". We will denote by $PD(attrName\langle OP\rangle attrValue)$ the padded string representation of the condition $attrName\langle OP\rangle attrValue$. We will represent by MAX and MIN the two strings of N characters each with the following property: each character of MAX has all its bits

[3] The reason why only the non-intensional attributes are considered will be explained later

to 1, and each character of MIN has all its bits to 0. This means that any comparison of an N-character string s to MAX will find s to be (lexicographically) smaller, and any comparison to MIN will find it larger. When a query q has a condition of the form $attrName\langle OP\rangle attrValue$, then its padded string representation is entered in the ITable along with the query q in the respective column. Given a class/individual o, the ITable is used to provide the list of queries with an inequality condition that is not satisfied by o. If o has an attribute attrName with a value attrValue, then we search on ITable for entries x that have Cond satisfying one of the following four specifications:

PD(attrName>attrValue) < x.Cond < PD(attrName>MAX)
PD(attrName≥attrValue) < x.Cond < PD(attrName≥MAX)
PD(attrName<MIN) < x.Cond < PD(attrName<attrValue)
PD(attrName≤MIN) < x.Cond < PD(attrName≤attrValue)

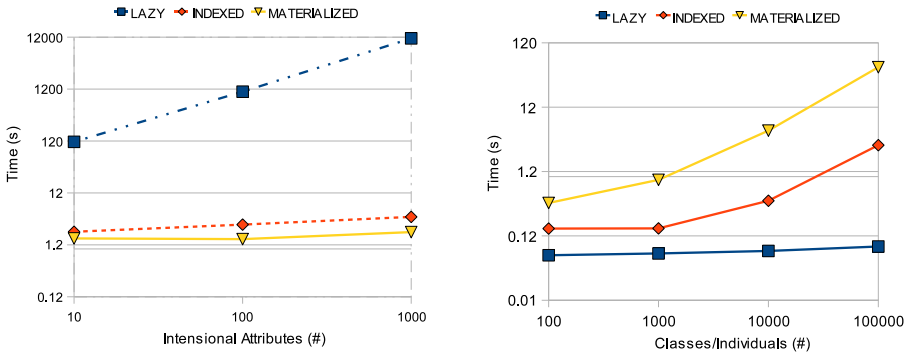
The queries found in this step are removed from the list of candidate queries $\mathcal{Q}$ that was generated in the process of equality conditions described earlier. What finally remains is the set of queries for which all their conditions are satisfied by the class/individual o that was inserted. Given these queries, one can find from the DTable the intensional attributes of o.

Example 4. Consider again the index structures illustrated in Figure 4, and the individual Canada of Figure 2. The ITable contains the inequality conditions of all queries in our intensional base example. From the equality process described in Example 3, we obtained the list of candidate queries for Canada: Q2, Q3 and Q5. Now, we need to eliminate from the candidate list those queries whose inequality conditions are not satisfied by the attributes of Canada. Therefore, for each (attribute, value) pair in Canada we create four requests to ITable, one for each range condition described above. For instance, for funding=70M, we create the following requests:

"___funding>00070" < x.Cond < "___funding>99999"
"___funding≥00070" < x.Cond < "___funding≥99999"
"___funding<00000" < x.Cond < "___funding<00070"
"___funding≤00000" < x.Cond < "___funding≤00070"

where 00000 and 99999 are the MIN and MAX of strings of length 5, respectively. The first request matches condition "___funding>00100" in ITable, which corresponds to Q12. The third request matches conditions "___funding<00010" and "___funding<00050", which correspond to Q11 and Q10, respectively. (Since there are only strict inequality conditions in ITable, the second and fourth requests match no attribute). Thus, Q10, Q11, and Q12 are queries whose inequality conditions are not satisfied and should be removed from the candidate list. However, since they are not in the list, no actual action is performed.

We repeat the same process for every attribute-value pair of Canada. When we perform the polling for population=32M, the ("population≤00000" < x.Cond < "population≤00032") matches the condition "population≤00020" in ITable, resulting in Q2 being removed from the candidate list. Since the candidate list at the end of the process is Q3 and Q5, we conclude that the individual Canada is in the domain of the intensional attributes: ⟨Q3,mustImplement,Q4⟩, and ⟨Q5,mustImplement,Q6⟩ ∎



(a) **(b)**

Fig. 5. Running times for finding (a) and inserting (b) intensional attributes

An extreme case is the one involving queries that have absolutely no constraints, and special consideration needs to be taken for them. Although we are taking care of such situations, we expect that this kind of queries will be extremely rare, since they simply assign an intensional attribute to every element that exists in the intensional base.

6 Experimental Evaluation

To evaluate the three implementation strategies, i.e., the lazy, the materialized and the indexed, we have conducted a number of experiments. We have repeated each experiment three times starting each time with a cold Java Virtual Machine (JVM). Our report time is the average of these three independent runs. The experiments were carried on a Windows XP machine powered by a 1.66 GHz CPU with 2 GB of RAM. The implementation utilizes the Protégé 3.3.1 plugin within the Eclipse environment. We used two different backends: the AllegroGraph 3.1 triple store [4] for the classes and individuals, and thus for the materialized implementation, and MySQL Server 5.0 [5] for the tables and the intensional definitions in the indexed and lazy implementations, respectively.

One of the basic operations we need to evaluate is the time required to find the intensional attributes of a give class/individual. Figure 5(a) shows the time required for this task for different sizes of the data. For this first set of experiments, we populated the knowledge base with 100000 classes/individuals and we tested the performance of our three approaches with three sets of intensional attributes: 10, 100 and 1000. The conclusion of the experiments were that the lazy approach does not scale well since it has to run the queries of all intensional attribute definitions, so the performance of this approach heavily depends on the number of intensional attributes stored in the system. In contrast, both materialized and indexed implementations have good performance regardless of the number of intensional attributes. There are different reasons for this in each optimized implementation. In the materialized approach, intensional attributes are translated to regular attributes, so the time in the graph corresponds to just finding

[4] http://agraph.franz.com/allegrograph
[5] http://www.mysql.com

the actual class/individual in the ontology storage. In the indexed approach, the time corresponds to evaluating simple queries in tables DTable, MTable, ETable and ITable, which are much smaller than the entire knowledge base. Note that the illustrated graphs here are in logarithmic scale.

In contrast to finding the intensional attributes, the task of inserting a new class/individual in the knowledge base has the opposite results, as shown in Figure 5(b). For this set of experiments, we used a fix number of 1000 intensional attributes and varied the number of classes/individuals in the knowledge base from 100 to 100000, as shown on the x axis. Note that any class/individual may have any number of intensional attribute definitions, hence the number of intensional definitions could be larger than the number of classes/individuals in the system (which is the case for the knowledge base of size 100 in the graph).

For an insertion using the materialized implementation, the times reported corresponds to evaluating all queries of the intensional attributes in order to find which ones to materialize in the new class/individual. This is essentially the same process the lazy approach performs for finding the intensional attributes of a given class/individual. The times reported for the insertion using the indexed implementation, on the other hand, corresponds to updating the information of the DTable, MTable, ETable and ITable, with the data of the queries in the new intensional attribute. Finally, the reported times for the lazy approach correspond to storing the queries of the new intensional attribute in the knowledge base, which requires just to store the intensional definition in the system.

7 Related Work

The idea of using queries for intensional definitions is not new. Derived concepts in Description Logics [4] are defined through logical expressions, i.e., queries. Derived elements in UML are also defined with some sort of logical expressions, although much simpler than those used in Description Logics. Virtual and materialized views in database management systems also use queries to describe their contents [7]. Queries as data values have been implemented in a number of commercial database systems such as INGRESS [5] and Oracle [13]. They have also been studied in the context of relational algebra [6] and Meta-SQL [8]. There have been numerous proposals for metadata management that include some kind of association between metadata and data, either by relating individuals values [14], subsets of the attributes in a tuple [15], or XML data with a complex structure [16]. The use of queries as data values for associating data and metadata has been studied in [9], where the authors propose a unified mechanism for modeling and querying across both data and metadata.

An ontology is a formal explicit description of concepts, or classes, in a domain of discourse [1]. An ontology, along with a set of individual instances, constitutes a knowledge base. In the context of the Semantic Web [17], ontologies are represented through formalisms like RDF [2] and OWL [11], and queried with ontology query languages such as SPARQL [12]. To the best of our knowledge, this is the first effort towards using queries to introduce intensional attributes in ontologies in order to tackle the issues described in this work.

8 Conclusion

We proposed an extension of the RDF and OWL formalisms with intensional attributes, i.e., attributes that have no explicit specification for the classes or individuals they associate (the domain and range of the attributes are specified through intensional expressions represented by queries). This work can be seen as an extension to attributes of the notion of derived concepts in Description Logics. Intensional attributes offer flexibility, great space and time savings, and can also be applied to future data. We investigated possible implementations and proposed one that provides good performance and space tradeoffs.

Acknowledgments: The current work has been partially supported by the EU grant GA-215032 and ICT-215874.

References

1. Noy, N.F., McGuinness, D.L.: Ontology development 101: A guide to creating your first ontology. Technical report, Stanford Knowledge Systems Laboratory KSL-01-05 (2001)
2. W3C: Resource description framework (RDF). http://www.w3.org/TR/rdf-concepts/ (2004)
3. W3C: RDF vocabulary description language 1.0: RDF Schema. http://www.w3.org/TR/rdf-schema/ (2004)
4. Baader, F., Nutt, W.: Basic Description Logics. In: Description Logic Handbook. (2003) 43–95
5. Stonebraker, M., Anton, J., Hanson, E.N.: Extending a Database System with Procedures. TODS **12**(3) (1987) 350–376
6. Neven, F., Bussche, J.V., Gucht, D.V., Vossen, G.: Typed Query Languages for Databases Containing Queries. In: PODS. (1998) 189–196
7. Lenzerini, M.: Data Integration: A Theoretical Perspective. In: PODS. (2002) 233–246
8. van den Bussche, J., Vansummeren, S., Vossen, G.: Towards practical meta-querying. Inf. Syst. **30**(4) (2005) 317–332
9. Srivastava, D., Velegrakis, Y.: Intensional Associations between Data and Metadata. In: SIGMOD. (2007) 401–412
10. Borgida, A., Brachman, R.J.: Modeling with Description Logics. In: Description Logic Handbook. (2003) 349–372
11. W3C: OWL web ontology language reference. http://www.w3.org/TR/owl-ref/ (2004)
12. W3C: SPARQL query language for RDF. http://www.w3.org/TR/rdf-sparql-query/ (2008)
13. Gawlick, D., Lenkov, D., Yalamanchi, A., Chernobrod, L.: Applications for Expression Data in Relational Database System. In: ICDE. (2004) 609–620
14. Buneman, P., Khanna, S., Tan, W.C.: On propagation of deletions and annotations through views. In: PODS, New York, NY, USA, ACM (2002) 150–158
15. Geerts, F., Kementsietsidis, A., Milano, D.: MONDRIAN: Annotating and querying databases through colors and blocks. In: ICDE. (2006) 82
16. Bertino, E., Castano, S., Ferrari, E.: On specifying security policies for web documents with an XML-based language. In: SACMAT. (2001) 57–65
17. W3C: Semantic web. http://www.w3.org/2001/sw/ (2008)