Knowledge representation in RDF/XML, KIF, Frame-CG and Formalized-English

Philippe Martin

Distributed System Technology Centre
Griffith University - PMB 50 Gold Coast MC, QLD 9726 Australia
philippe.martin@gu.edu.au

Abstract. This article shows how RDF/XML, KIF, Frame-CG (FCG) and Formalized-English (FE) can be used in a panorama of knowledge representation cases. It highlights various inadequacies of RDF/XML, advantages provided by high-level expressive notations (FCG and FE), and the KIF translations provide a logical interpretation for the other notations. Knowledge providers may see this document as a guide for knowledge representation. Developers may see it as a list of cases to take into account for their notations and inferences engines. We are working to make our tool, WebKB-2, import and export in these notations.

1 Introduction

A knowledge-based system (KBS) generally uses only one *model* to store and exploit knowledge, e.g. a semantic network model such as Conceptual Graphs (CGs) [1] or the Resource Description Format (RDF) [4], but may import/export (i.e. accept/present) representations in various *notations*, e.g. KIF (Knowledge Interchange Format) [2] or RDF/XML (the XML linearization of RDF) [4]. Are some notations better than others for knowledge representation and exchange?

Expressivity is one criteria. Indeed, the more a notation has "features", the more information can be entered precisely and then exploited for inferencing and exchanged (since the "keywords" for the features are standard). As is usual in the KIF/RDF/XML worlds, an inference engine is not obliged to take into account all the features of a notation and perform all the logical deductions. What inferencing is done is an application choice. Hence, the issues of completeness and decidability are *not* related to notations but to inference engines. A limited notation such as RDF/XML seriously and arbitrarily limits knowledge sharing and usability.

Conciseness and *readability* are other desirable criterias for a notation since they ease the understanding of knowledge (and for developers, ease implementation and debugging). Knowledge entering easiness is a criteria related to conciseness and how *high-level* the notation is, i.e. how many ontological distinctions have a special intuitive syntax. Also related and important is the *knowledge normalizing effect* of the notation: the fewer choices a knowledge provider has for representing a piece of information, the more easily and safely this representation can be automatically related and compared to other ones. Thus, high-level expressive notations seem better for knowledge representation and exchange than low-level expressive notations such as KIF, or low-level restricted notations such as RDF/XML.

In the Semantic Web community, although the need for higher-level notations than RDF/XML is recognized[1], XML is most often used as linearization format. For example, Tim Berners-Lee writes that his Notation3 is "not as an alternative to RDF's XML syntax which has the advantage that it is in XML"[2]. One may wonder what this advantage is since he also acknowledges that most notations may be "web-ized"[3] by using URIs for category identifiers. Even if knowledge can be represented in XML, it is unlikely that XML objects are directly used by advanced inference engines, and that knowledge providers read or write XML-based languages. Hence, translations to and from XML are necessary and likely to be the cause of information loss and errors in softwares. From a syntactical viewpoint, the use of a Lisp-based notation (e.g. KIF) as a general low-level interlingua makes more sense because Lisp is concise and has adequate quotation (i.e contextualization) features. From any viewpoint we can think of, the use (and ideally, the standardization) of a high-level expressive notation makes even more sense since then knowledge is easier to write, read, compare, exchange and exploit. Being readable and not XML-based, knowledge representations can also be mixed and hyperlinked with text and images within HTML/XML documents (our tools WebKB-1[8] and WebKB-2[10] exploit such documents).

From the knowledge provider's viewpoint, the main problem is how to express knowledge. Documentations about notations often only provide a grammar, a few simple examples, and omit to explain how to represent more complex cases commonly found in natural language sentences, or to state that some of these cases cannot be represented. For example, the RDF/XML documentation [4] is currently very poor. The documentation of KIF is completed by the Ontolingua library[4] but several knowledge representation cases are still difficult to find.

This article presents a panorama of knowledge representation features[5] and shows how various notations can be used (or extended to be used) to cover these features. In addition to RDF/XML and KIF, this document presents two notations derived from CGLF [11], and designed to be as intuitive[6] as possible in all the presented cases: Formalized English (FE) and Frame-CG (FCG) [9] [7]. Knowledge providers may see this document as a guide for knowledge representation. KBS developers may see it as a list of cases to take into account. Language developers

[1] http://www.w3.org/DesignIssues/Logic.html
[2] http://www.w3.org/DesignIssues/Notation3.html
[3] http://www.w3.org/DesignIssues/RDFnot.html
[4] http://www-ksl-svc.stanford.edu:5915/
[5] Only "logical features"; see http://www.webkb.org for ontological examples.
[6] The use of English articles or expressions as (extended) quantifiers, one of our ideas to obtain a more intuitive and "knowledge normalizing" notation, was also applied (although to a less extent) in KM, the Knowledge Machine notation [3].
[7] WebKB-1, our first Web-based KBS, imports and exports CGLF, FCG and FE. WebKB-2 [10] currently only uses FCG and partially exports in RDF/XML, but will later also import and export in FE, CGIF and KIF. Grammars and parsing examples of these notations are at: http://www.webkb.org/doc/grammars/

may see it as a workbench for comparing their notations to others. The KIF translations also provide logical interpretations for the other notations.

In the RDF/XML example of this article, we follow the RDF lexical conventions (intercap style, first letter of class identifiers in uppercase)[8] and the RDFS [5] and DAML+OIL [6] schemas. For the other notations, we follow more appropriate lexical conventions (e.g. singular nouns, English spelling) and ontological conventions that we have advocated in [9] for knowledge comparison, retrieval and exchange. Except in Section 11 (which deals with category declaration) the categories used in the examples are supposed to be declared (and, for the RDF representations, declared in the same RDF document).

2 Conjunctive Existentially Quantified Sentences

Here is an example of such simple forms of knowledge. "E" is for "English".

```
E:    Tom owns a dog that is not Snoopy.
FE:   Tom is owner of a dog different_from Snoopy.
FCG:  [Tom, owner of: (a dog != Snoopy)]
KIF:  (exists ((?x dog)) (and (owner ?x Tom) (/= ?x Snoopy)))
RDF:  <Dog><owner><rdf:Description about="#Tom"/></owner>
           <daml:differentIndividualFrom resource="#Snoopy"/></Dog>
```

3 Contextualization

Contexts allow us to represent statements over statements. Contexts are represented via delimitors (e.g. '...' in FE, [...] in FCG, '(...) and ^(...) in KIF) or keywords (e.g. aboutEach in RDF).

```
E:    Tom believes Mary now likes him (in 2002) and before she did not.
FE:   Tom is believer of ' *p 'Mary is liking Tom' at time 2002'
      and is believer of '!*p is before 2002'.
FCG:  [Tom, believer of: [*p [Mary, agent of:(a liking,object:Tom)], time:2002],
            believer of: [!*p, before: 2002] ]
KIF:  (exists (?p)
        (and (= ?p '(exists ((?x liking)) (and (agent *l Mary) (object ?l Tom))))
             (believer ^(time ,?p 2002) Tom)   //',?p'->the value of ?p is quoted
             (believer ^(before (not ,?p) 2002) Tom)))
RDF: <rdf:Description bagID="p">
          <Liking><agent><rdf:Description about="#Mary"/></agent>
                  <object resource="#Tom"/></Liking></rdf:Description>
     <rdf:Description bagID="not_p" aboutEach="#p"  truth ="false"/>
     <rdf:Description bagID="now"   aboutEach="#p"  time  ="2002"/>
     <rdf:Description bagID="past"  aboutEach="#not_p"  before="2002"/>
     <rdf:Description aboutEach="#now"><believer resource="#Tom"/>
     </rdf:Description>
     <rdf:Description aboutEach="#past"><believer resource="#Tom"/>
     </rdf:Description>
```

[8] http://www.w3.org/TR/REC-rdf-syntax/#usage

A problem with the above RDF sentence is that the `truth` property was invented by [7] but is not in the RDF/RDFS standard nor in DAML+OIL.

Relations of type `believer`, `time` and `before` connect an instance of the type `situation` to another object. In CGs, it is customary to distinguish the "proposition" stated by a sentence/graph/formula from the "situation" described by this proposition. However, making this distinction is sometimes difficult for inexperienced people, and it is inconvenient because it leads to adding several intermediary contexts. Since these intermediary contexts can be automatically inserted by a parser (thanks to the signatures of the used relations), we have not included these intermediary contexts in the above example. (We also assumed parsers can understand that `2002` is a date, again thanks to relation signatures).

In FE and FCG, variables may be prefixed by '?' or '*' (or '@' for collections, as in KIF). When a variable is first used in a graph, it must be associated with a type and a quantifier. When, within a graph, a variable re-use exists in a context (c1) different from the context (c2) where the variable has been introduced, the convention is that the variable is assumed to have been introduced in the minimum upper context embedding c1 and c2.

FE and FCG also permit the introduction of *free variables* with the prefix '^'. Their semantics are the same as in KIF: within statements (as opposed to queries), these variables are assumed to be introduced and universally quantified in some upper context (again, the lowest context that includes all the introductions and re-uses of the variables).

4 Universal Quantification

```
E:    Animals have exactly one head.
FE:   Any animal has for part 1 head.
FCG:  [any animal, part: 1 head]
KIF:  (forall ((?a animal)) (exists1 ?h (and (head ?h) (part ?a ?h))))
RDF:  <rdf:Property ID="headPart">
        <rdfs:subPropertyOf resource="#part"/><rdfs:range resource="#Head"/>
      </rdf:Property>
      <rdfs:Class rdf:about="#Animal">
        <rdfs:subClassOf><daml:Restriction daml:cardinality="1">
                           <daml:onProperty resource="#headPart"/>
                         </daml:Restriction></rdfs:subClassOf></rdfs:Class>
```

Here is our KIF definition of `exists1`:

```
(defrelation exists1 (?var ?predicate) :=
  (truth ^(exists (,?var) (and (,?predicate ,?var)
                  (forall (?y) (=> (,?predicate ?y) (= ,?var ?y)))))))
```

The RDF representation was derived from an example in the DAML+OIL documentation. However, this representation uses a *category definition* (for `Animal`) instead of a *sentence* with a universal quantifier. As explained in the KIF standard [2], "definitions have no truth values in the usual sense; they are so because we say that they are so". Because RDF and RDFS plus DAML+OIL have no universal quantifier and because definitions are easier to handle than universal quantifier, it is best to overlook this subtle distinction in the RDF representation. A more general construct is proposed in the next section.

Another problem of this RDF representation is that it contains a declaration of the relation `headPart`. We would have liked to use a lambda abstraction (i.e. an anonymous category definition) instead but `Restriction` can only be used for defining an anonymous concept type, not an anonymous relation type.

5 Lambda Abstraction, Percentage, Possibility, Valuation

```
E:    At least 93% of healthy birds can fly.
FE:   At least 93% of [bird with chrc a good health] can be agent of a flight.
FCG:  [at least 93% of (bird, chrc: a good health), can be agent of: a flight]
KIF:  (defrelation healthy_bird (?b) :=
         (exists ((?h health)) (and (bird ?b) (chrc ?b ?h) (measure ?h good))))
      (forAtLeastNpercent 93 ?x healthy_bird
         (exists ((?f flight)) (physical_possibility (agent ?f *x))))
RDF:  <forall var="#b" at_least percent="93%">
        <exists var="h">
          <if><Bird about="#b"><chrc><Health about="#h"><measure resource="#good"/>
                                  <Health></chrc></Bird>
            <then><exists var="f">
                    <Flight about="#f"><agent can resource="#b"/></Flight>
                  </exists></then></if></exists></forall>
```

Here is our KIF definition of `forAtLeastNpercent` (and associate functions):

```
(defrelation forAtLeastNpercent (?n ?var ?type ?predicate) :=
  (exists ((?s set))
    (and (truth ^(forall (,?var) (=> (member ,?var ,?s) (,?type ,?var)))
         (>= (numMembersSuchThat ,?s ,?predicate) (/ (* (size ,?s) ?n) 100)))))

(define-function numMembersSuchThat (?set ?p) :-> ?num :=
  (if (and (set ?set) (predicate ?p)) (numElemsSuchThat (listOf ?set) ?p)))


(define-function numElemsSuchThat (?list ?p) :-> ?num
  (cond ((null ?list) 0)
        ((list ?list) (if ?p (1+ (numElemsSuchThat (rest ?list) ?p))))))
```

We have not found a simple way to represent a lambda-abstraction (that is, an anonymous type declaration) in KIF. Hence, we have used a normal type declaration.

The above "RDF" sentence actually does not follow the RDF/XML grammar. It re-uses the `forall` and `exists` constructs that were invented by Berners-Lee in 1999 [7] to represent universal quantification but which were not included in RDF. We added the attributes `percent` and `at_least` to the `forall` construct. The insertion of the keyword/attribute `can` into a relation is also an extension of RDF and RDF/XML.

The above example can be modified to refer to "most birds" instead of "93% of birds". In FE and FCG, the keyword `most` may be used. It is equivalent to `at least 60%` (hence, it can be represented in KIF in this form).

6 Negations, Exclusions and Alternatives

We have already seen two forms of negation: one involving a `different_from` relation (`differentIndividualFrom` in DAML+OIL, `/=` in KIF), and one involving the negation of a sentence ("not" in KIF). This last form is more difficult to exploit by inference engines and leaves room for ambiguity. For example, "Tom does not own a blue car" may mean that "Tom has a car but not blue" or "Tom does not have a car". Thus, it is better to use the first form, or break sentences into smaller blocks connected by coreference variables to reduce or avoid ambiguities.

Here is a variant of the first form: negation on types.

```
E:    Tom owns something that is not a car.
FE:   Tom is owner of a !car.
FCG:   [Tom, owner of: a !car]
KIF:  (exists (?type ?x) (and (owner ?x Tom) (holds ?type ?x) (/= ?type car)))
RDF:  <rdf:Description ID="x">
         <owner><rdf:Description about="#Tom"/></owner>
         <rdf:type><rdf:Description ID="aType">
                      <daml:differentIndividualFrom resource="car"/>
                  </rdf:Description></rdf:type></rdf:Description>
      <!-- "disjointWith" may be used instead of "differentIndividualFrom" -->
```

Exclusion between objects (and hence, some forms of negation) may also be represented via collections of exclusive objects. RDFS proposes an `alt` collection to store alternatives but unfortunately does not specify if this "or" is inclusive or exclusive. Specializing `alt` by `or_bag` and `xor_set` seems a good idea even if RDF parsers are unlikely to take advantage of this distinction. However, the current RDF/XML grammar only permits to define members (using `li` relations) to collections of types `Bag`, `Alt` and `Seq`, not specializations of these types.

FE and FCG use OR-collections and XOR-collections as a syntactic mean to store "or" and "xor" relations between objects (types, instances or sentences). This can be done in RDF too (however, to store exclusion relations between categories, it is better to use the DAML relations `disjointWith`, `complementOf` and `inverseOf`). Here is an example of OR-collection between instances. (Note: `red`, `yellow` and `orange` are not instance but subtype of `color`, and have many subtypes, e.g. `crimson`, `dark_red` and `chrome_red`. Their instances are the actual occurrences of color that physical objects have.)

```
E:    Tom's car is red, yellow or orange.
FE:   Tom is owner of a car that has for color OR{a red, a yellow, an orange}.
FCG:  [Tom, owner of: (a car, color: OR{a red, a yellow, an orange})]
KIF:  (exists ((?x car) ?c)
        (and (owner ?x Tom) (color ?x ?c) (or (red ?c)(yellow ?c)(orange ?c))))
RDF:  <Car><owner><rdf:Description about="#Tom"/></owner>
           <color><rdf:Description>
                    <rdf:type><rdf:Alt><rdf:li resource="#Red"/>
                                       <rdf:li resource="#Yellow"/>
                                       <rdf:li resource="#Orange"/></rdf:Alt>
                    </rdf:type></rdf:Description></color></Car>
```

In this example, it would have been simpler to use a type such as `warm_color` instead of the OR-collection of `red`, `yellow` and `orange` (and this form makes inferencing easier). More generally, this section shows that a negation can be represented in numerous ways and that these representations are difficult for an inference engine to compare and hence exploit fully. Both for knowledge exchange with frame-based systems and for knowledge inferencing, `different_from` relations between categories should be prefered to other forms of negations.

7 Collections and Quantifier Precedence

Collections have been introduced in the previous section and via examples using numerical quantifiers. In this section, we show how various interpretations of the English sentence *"4 persons have approved 3 resolutions"* (and some variations of it) can be interpreted. By studying how to represent relations between members of two simple collections, we illustrate the importance of specifying how a collection must be interpreted, and show how to handle complex cases of quantifier precedence (between numerical, existential and universal quantifiers).

The sentence *"4 persons have approved 3 resolutions"* is ambiguous. The 4 persons may have individually or collectively approved 3 resolutions (the same 3 or not), and "collectively" may have two meanings: the participation in a "unique" approval act or the approval of "most" of the resolutions (or a combination of both as illustrated in the last example of this section). In this paper, "persons acting together/collectively" means that "there exists an act and each of the persons is an agent of that act". This interpretation of "collectiveness" was used by Sowa in [11] and, in CG terminology, it implies that the act must be represented by a concept node, not by a relation node (this has not been made explicit by Sowa).

In CGs [11], any collection in a concept node of a CG can be specified as having a *distributive interpretation* (each member of the collection individually participates to the relations associated with the node), a *collective interpretation* (the members collectively participate in the relations associated with the node), a *defaut interpretation* (an unspecified mix of collective and distributive interpretation) or a *cumulative interpretation* (the relations are about the collection itself).

RDF permits the distributive interpretation (or is it the default interpretation?) via the keyword `aboutEach`. Without this keyword, relations are about the collection itself (cumulative interpretation). However, the authors of RDF also represent the collective interpretation via direct relations to a bag (see Section 3.5 of [5]). This ambiguity can often be repaired by RDF parsers thanks to the signatures of the relations (if the expected type is not a collection, this is the collective interpretation, otherwise the ambiguity remains). Hence, we have not introduced a new keyword for specifying the collective interpretation in the following examples. However, we have introduced the concept type `set` and the relation type `size` since neither RDF, RDFS nor DAML+OIL propose equivalent categories. The type `set` is declared as a subtype of `rdf:Bag` to permit the use of `aboutEach`.

The first example keeps the ambiguity of the above sentence (both collections have the default interpretation). The 's' at the end of `persons` and `resolutions` in the FE and FCG representations are supposed to be automatically removed (WebKB-2 does this when a numerical or universal quantifier is involved). To highlight the logical interpretations, this section provides predicate logic (PL) translations instead of FE translations.

```
E:    4 persons have (each/together) approved 3 resolutions.
FCG:  [4 persons, agent of: (an approval, object: 3 resolutions)]
KIF:  (forAllN 4 ?p person (forAllN 3 ?r resolution
         (exists ((?a approval)) (and (agent ?a ?p) (object ?a ?r)))))
RDF:  <Set ID="Persons"><size>4</size></Set>
      <forall var="p">
        <if><Person about="#p"/><memberOf resource="#Persons"/></Person>
        <then><Set ID="Rs"><size>3</size></Set>
              <forall var="r">
                <if><Resolution about="#r"><memberOf resource="#Rs"/></Resolution>
                <then><exists var="a">
                        <Approval about="#a"><agent  resource="#p"/>
                                             <object resource="#r"/></Approval>
                      </exists></then></if></forall></then></if></forall>
```

PL: $\exists ps\ set(ps) \wedge size(ps,4) \wedge \forall p \in ps\ \ \exists rs\ set(rs) \wedge size(rs,3) \wedge \forall r \in rs$
$\exists a\ approval(a) \wedge agent(a,p) \wedge object(a,r)$

For modularity, we have introduced the "quantifier" `forAllN`.

```
(defrelation forAllN (?num ?var ?type ?predicate) :=
  (exists ((?s set)) (and (size ?s ?num)
     (truth ^(forall (,?var) (=> (member ,?var ,?s)
                                 (and (,?type ,?var) ,?predicate)))))))
```

In FE and FCG, the order and scope of the quantifiers follow the order and structure of the graphs. The RDF standard does not specify how to delimit order and scope of quantifiers because the only direct form of universal quantification it has is via RDF representations using `aboutEach`. The next example shows a simple inversion of the quantifier scopes.

```
E:    3 resolutions have been approved by 4 persons (each/together).
FCG:  [3 resolutions, object of: (an approval, agent: 4 persons)]
KIF:  (forAllN 3 ?r resolution (forAllN 4 ?p person
         (exists ((?a approval)) (and (agent ?a ?p) (object ?a ?r)))))
RDF:  <Set ID="Rs"><size>3</size></Set>
      <forall var="r">
        <if><Resolution about="#r"/><memberOf resource="#Rs"/></Resolution>
        <then><Set ID="Persons"><size>4</size></Set>
              <forall var="p">
                <if><Person about="#p"><memberOf resource="#Persons"/></Person>
                <then><exists var="a">
                        <Approval about="#a"><agent  resource="#p"/>
                                             <object resource="#r"/></Approval>
                      </exists></then></if></forall></then></if></forall>
```

PL: $\exists rs\ set(rs) \wedge size(rs,3) \wedge \forall r \in rs\ \ \exists ps\ set(ps) \wedge size(ps,4) \wedge \forall p \in ps$
$\exists a\ approval(a) \wedge agent(a,p) \wedge object(a,r)$

In FE and FCG, the *collective interpretation* is specified via the keywords `together`, `group of`, `set of`, `bag of`, `list of`, `sequence of` or `alternatives` (the first three are synonyms; in this paper, we most often use `set` to save space).

If we take the two previous examples and gradually introduce the collective interpretation for the collections, we obtain five different logical interpretations (instead of six because when both collections are collectively interpreted, the inversion of quantifier scopes does not change the meaning). Below are three of these combinations (the other two are: "A group of 3 resolutions has been approved by 4 persons" and "A group of 4 persons has approved 3 resolutions"). In the third case, we give the three equivalent FCG sentences and a tentative RDF representation using `aboutEach`. For the other cases, using `aboutEach` is not possible, the `forall` and `exists` constructs have to be used (in the same order as in PL).

```
E:    4 persons have (each/together) approved a group of 3 resolutions.
FCG:  [4 persons, agent of: (an approval, object: a set of 3 resolutions)]
KIF:  (forAllN 4 ?p person (exists ((?rs set) (?a approval))
        (forAllIn ?rs 3 ?r resolution (and (agent ?a ?p) (object ?a ?r)))))
```

PL: $\exists ps\ set(ps) \wedge size(ps, 4) \wedge \forall p \in ps\ \ \exists rs\ set(rs) \wedge size(rs, 3) \wedge$
$\exists a\ approval(a)\ \forall r \in rs\ agent(a, p) \wedge object(a, r)$

```
E:    3 resolutions have been approved by a group of 4 persons.
FCG:  [3 resolutions, object of: (an approval, agent: a set of 4 persons)]
KIF:  (forAllN 3 ?r resolution (exists ((?ps set) (?a approval))
        (forAllIn ?ps 4 ?p person (and (agent ?a ?p) (object ?a ?r)))))
```

PL: $\exists rs\ set(rs) \wedge size(rs, 3) \wedge \forall r \in rs\ \ \exists ps\ set(ps) \wedge size(ps, 4) \wedge$
$\exists a\ approval(a)\ \forall p \in ps\ agent(a, p) \wedge object(a, r)$

```
E:    A group of 4 persons has approved a group of 3 resolutions.
FCG:  [a set of 4 persons,agent of:(an approval,object:a set of 3 resolutions)]
 or:  [a set of 3 resolutions,object of:(an approval,agent:a set of 4 persons)]
 or:  [an approval, agent: a set of 4 persons, object: a set of 3 resolutions]
KIF:  (exists ((?r approval) (?ps set) (?rs set))
        (forAllIn ?ps 4 ?p person (forAllIn (?rs 3 ?r resolution
          (and (agent ?a ?p) (object ?a ?r))))))
RDF:  <Set ID="Persons"><size>4</size></Set>
      <Set ID="Resolutions"><size>3</size></Set>
      <rdf:Description aboutEach="#Persons"> <rdf:type resource="#Person"/>
        <agentOf><Approval><object><rdf:Description aboutEach="#Resolutions"/>
                          </object></Approval></agentOf></rdf:Description>
```

PL: $\exists a\ approval(a) \wedge \exists ps\ set(ps) \wedge size(ps, 4) \wedge\ \ \exists rs\ set(rs) \wedge size(rs, 3) \wedge$
$\forall p \in ps\ \forall r \in rs\ agent(a, p) \wedge object(a, r)$

Here is how we define the "quantifier" `forAllIn`.

```
(defrelation forAllIn (?s ?num ?var ?type ?predicate) :=
  (and (size ?s ?num) (truth ^(forall (,?var) (=> (member ,?var ,?s)
                                          (and (,?type ,?var) ,?predicate))))))
```

The RDF/XML syntax is so poor that it requires the use of a type `agentOf` which must be defined as inverse of `agent`. This is un-natural and time-consuming for the user, and imposes additional work to inference engines.

In FE and FCG, the *distributive interpretation* is specified via the keyword `each`. If we introduce the collective interpretation into the previous seven combinations, we obtain nine different logical interpretations. Here are two of them (again, the RDF/XML representations would have to use `forall` and `exists` constructs in the same order as in the KIF or PL representations).

```
E:    4 persons have each approved 3 resolutions.
FCG:  [each of 4 persons, agent of: (an approval, object: 3 resolutions)]
KIF:  (forAllN 4 ?p person (exists1For ?p ?rs set (forAllIn ?rs 3 ?r resolution
         (exists1For ?p ?a approval (and (agent ?a ?p) (object ?a ?r))))))
```

PL: $\exists ps\ set(ps) \wedge size(ps,4) \wedge \forall p \in ps\ \ \exists!!rs\ set(rs) \wedge size(rs,3) \wedge \forall r \in rs$
$\exists!!a\ approval(a) \wedge agent(a,p) \wedge object(a,r)$

```
E:    4 persons have each approved a group of 3 resolutions.
FCG:  [each of 4 persons,agent of:(an approval,object:a set of 3 resolutions)]
KIF:  (forAllN 4 ?p person (exists1For ?p ?rs set (exists1For ?p ?a approval
         (forAllIn ?rs 3 ?r resolution (and (agent ?a ?p) (object ?a ?r))))))
```

PL: $\exists ps\ set(ps) \wedge size(ps,4) \wedge \forall p \in ps\ \ \exists!!rs\ set(rs) \wedge size(rs,3) \wedge$
$\exists!!a\ approval(a)\ \forall r \in rs\ agent(a,p) \wedge object(a,r)$

Below is our KIF definition of `exists1For` ($\exists!!$). This quantifier permits us to specify that the persons are agent of different approvals and different resolutions (first example above) or groups of resolutions (second example above).

```
(defrelation exists1For (?var1 ?var2 ?type ?predicate) :=
    (truth ^(exists (,?var2)
       (and (,?type ,?var2) (,?predicate ,?var1 ,?var2)
            (forall (?x) (=> (,?predicate ,?var1 ?x) (= ,?var2 ?x)))
            (forall (?y) (=> (,?predicate ?y ,?var2) (= ,?var1 ?y)))))))
```

Finally, we can also introduce "most" as an interpretation of collectiveness in each of the previous (7+9=16) combinations (hence, 16 logical interpretations again). Here is one of them.

```
E:    A group of 3 resolutions has been approved by most in a group of 4 persons.
FCG:  [a group of 4 persons, agent of:
            (an approval, object: most in a group of 3 resolutions)]
 or:  [most in a group of 3 resolutions, object of:
            (an approval, agent: a group of 4 persons)]
KIF:  (exists ((?r approval) (?ps set) (?rs set))
         (forAllIn ?ps 4 ?p person (forMostIn (?rs 3 ?r resolution
           (and (agent ?a ?p) (object ?a ?r))))))
```

PL: $\exists a\ approval(a)\ \wedge\ \exists ps\ set(ps) \wedge size(ps,4)\ \wedge\ \ \exists rs\ set(rs) \wedge size(rs,3)\ \wedge$
$\forall p \in ps\ agent(a,p)\ \wedge\ \exists mostOfrs\ set(mostOfrs)$
$(\forall r \in rs\ (object(a,r) => r \in mostOfrs))\ \wedge\ size(mostOfrs) >= 2$
$//\ >= 2\ since\ size(rs)/2 = 1.5$

Here is how we define `forMostIn`.

```
(defrelation forMostIn (?set ?num ?var ?type ?predicate) :=
  (and (size ?set ?num)
       (truth ^(forall (,?var) (=> (member ,?var ,?set) (,?type ,?var))))
       (>= (numMembersSuchThat ,?set ,?predicate) (* (size ,?set) 0.60))))
```

8 Intervals

```
E:    Tom has been running for 45 minutes to an hour.
FE:   Tom is agent of a run with duration a period with part 45 to 60 minutes.
FCG:  [Tom, agent of: (a run, duration: (a period, part: 45 to 60 minutes))]
KIF:  (exists ((?r run) (?p period) (?minutes set))
        (and (agent ?r Tom) (duration ?r ?p)
             (forAllIn ?minutes 45 60 ?m minute (part ?p ?m))))
RDF:  <Run><agent resource="#Tom"/><duration><Period ID="p"/></Run>
      <Set ID="Minutes"><minSize>45</minSize> <maxSize>60</maxSize></Set>
      <rdf:Description aboutEach="#Minutes">
        <rdf:type resource="#Minute"/> <partOf resource="#p"></rdf:Description>
```

Here is how we define `forAllInBetween`.

```
defrelation forAllInBetween (?s ?n1 ?n2 ?var ?type ?predicate) :=
  (exists (?n) (and (size ?s ?n) (>= ?n ?n1) (=< ?n ?n2)
    (truth ^(forall (,?var) (=> (member ,?var ,?s)
                                (and (,?type ,?var) ,?predicate))))))
```

In all these notations, a concept node of type `period` had to be introduced since the minutes participate in the same period/duration. This is the same problem as the collective participation to an act: the act cannot be represented as a relation node. Here, a relation of type `duration` cannot directly connect the run to the minutes. However, we only became aware of this problem when trying to produce the KIF representation.

9 Function Calls and Lists

Special syntactic sugar to distinguish functional relations from other relations is not mandatory since this distinction can be specified in the relation type declaration (hence, we assume that all notations permit function calls even if they do not permit function definitions). However, a syntactical difference eases readablility and syntactic checking. The following example involve two functions (`length, +`) and one relation (<).

```
E:    The length of the list "Tom, Joe, Jack" plus 1 is less than 5.
FE:   length(LIST{Tom,Joe,Jack}) + 1 < 5.
FCG:  [length(LIST{Tom,Joe,Jack}) + 1 < 5]
KIF:  (superior (+ (length (listof Tom Joe Jack)) 1) 5)
RDF:  <rdf:Seq><rdf:li resource="#Tom"/><rdf:li resource="#Joe"/>
                <rdf:li resource="#Jack"/>
                <length><Number ID="l"><length></rdf:Seq>
      <plus> <arg1 resource="#l"> <arg2>1</arg> <arg3><Number/></arg3> </plus>
```

A problem with the RDF/XML notation is that `length, plus, arg1, arg2` and `arg3` are not declared in RDF/RDFS/DAML+OIL.

In FE and FCG, the notation for functional relations can also be used for representing relations which are not binary.

10 Higher-order Statements

Higher-order statements are needed to quantify over types. For example, a first-order statement can describe the transitivity of a particular relation (e.g. "the part of a part is also a part") but a second-order statement is required for describing in general what a transitive relation is. Since definitions will be presented in the next section, the next example uses the characteristic `transitivity` instead of defining a type such as `transitive_binary_relation`.

```
E:    If a binary relation type rt is transitive
      then if x is connected to y by a relation of type rt, and
              y is connected to z by a relation of type rt,
           then x is connected to z by a relation of type rt.
FE:   If ‘a binaryRelationType ^rt has for chrc the transitivity’
      then ‘if ‘^x has for ^rt ^y that has for ^rt ^z’
            then ‘^x has for ^rt ^z’ ’.  //rt,x,y,z are free variables
FCG:  [ [a binaryRelationType ^rt, chrc: the transitivity] =>
        [ [^x, ^rt: (^y, ^rt: ^z)] => [^x, ^rt: ^z] ] ]
KIF:  (exists ((?t transitivity))
        (forall ((?rt binaryRelationType) ?x ?y ?z)
          (=> (chrc ?rt ?t)
            (=> (and (holds ?rt ?x ?y) (holds ?rt ?y ?z)) (holds ?rt ?x ?z)))))
RDF:  <forall var="r" v2="x" v3="y" v4="z"> <!-- v2,v3,v4: new attributes -->
        <if><rdf:Description about="#rt"><chrc><transitivity/></chrc>
            </rdf:Description>
        <then><if><rdf:Description about="#x">
                    <rdf:property pname="#rt"><!-- pname: new attribute -->
                      <rdf:Description about="#y">
                        <rdf:property pname="#rt"><rdf:Description about="#z"/>
                        </rdf:property></rdf:Description>
                    </rdf:property></rdf:Description>
              <then><rdf:Description about="#x">
                      <rdf:property pname="#rt"><rdf:Description about="#z"/>
                      </rdf:property></rdf:Description>
              </then></if></then></if></forall>
```

To permit a variable to refer to a relation type, the RDF sentence uses another extension to XML from [7]: the `pname` attribute. The additional variable names `v2, v3, v4` in the `forall` construct also come from Berners-Lee.

The FE and FCG representations need not use universal quantifiers: the location of the re-use of the relation type `rt` (i.e. within a relation node) specifies there is a mapping from `rt` to a free variable referring to a relation of type `rt`.

11 Declarations and Definitions

In RDF/XML, a category is uniquely identified by a URI, e.g. `http://www.foo.com` and `http://www.bar.com/doc.html#car`. In a multi-user KBS such as WebKB-2 [10], user identifiers are more convenient knowledge source identifiers than document URIs. Thus, in WebKB-2, a category identifier may be a URI, an e-mail address or the concatenation of a source identifier and a key name, e.g. `wn#dog` and `pm#IR_system` ("wn" refers to WordNet 1.7 and "pm" is the login name of the user represented by the category `philippe.martin@gu.edu.au`). In this third case, the category may still be referenced from outside the KB by prefixing the identifier with the URL of the KB, e.g. `http://www.webkb.org/kb/wn#dog`.

This identifier encoding is used for all the input/output notations in WebKB-2 (FCG, FE, KIF, CGIF) except for RDF/XML where URIs have to be used.

In addition to an *identifier*, a category may have various *names* (which may also be names of other categories). In FE and FCG, a category identifier may *show* several names, e.g. `wn#dog__domestic_dog__Canis_familiaris` (at least two underscores must be used for separating the names).

WebKB-2 proposes a special notation for declaring categories and links (i.e. second-order relations) between them: the "For Ontology" (FO)[9] notation. It is an extension of the special notation used in CGs for specialization links between types. Hence, in the following example, we use FO instead of FE and FCG.

For the RDF representation, the categories created by the user "pm" are supposed to be declared in the same document (otherwise each referred resource must be prefixed by "http://www.webkb.org/kb/pm#").

For the KIF representation, we have chosen to use relation types from RDF, RDFS and DAML+OIL (rather than from the Frame-ontology and OKBC-ontology of the Ontolingua library) to ease the comparison with RDF/XML representations.

```
FO:  pm#thing__top_concept_type (^thing that is not a relation^) 29/11/1999
       _  chose (oc fr),
       ^  rdfs#class,    !  pm#relation,    =  sowa#T,
       >  {(pm#situation pm#entity)} pm#thing_playing_some_role;
KIF: (defrelation pm#thing ()) (rdfs#class pm#thing)
       (pm#name pm#thing "thing") (pm#name pm#thing "top_concept_type")
       (pm#nameWithCreatorAndLanguage pm#thing "chose"
                        Olivier.Corby@sophia.inria.fr rdfs#class)
       (dc#Creator pm#thing philippe.martin@gu.edu.au)
       (dc#Date pm#thing "29/11/1999")
       (rdfs#comment pm#thing "thing that is not a relation")
       (daml#disjointWith pm#thing pm#relation)   (= pm#thing sowa#T)
       (daml#disjointUnionOf pm#thing '(pm#situation pm#entity))
       (rdfs#subClassOf pm#thing_playing_some_role pm#thing)
RDF: <rdfs:Class rdf:about="http://www.webkb.org/kb/pm#Thing">
       <rdfs:label>thing</rdfs:label> <rdfs:label>top_concept_type</rdfs:label>
       <rdfs:label xml:lang="fr"
                 dc:creator="Olivier.Corby@sophia.inria.fr">chose</rdfs:label>
       <dc:Creator>philippe.martin@gu.edu.au</dc:Creator>
       <dc:Date>29/11/1999</dc:Date>
       <rdfs:comment>thing that is not a relation</rdfs:comment>
       <rdf:type resource="http://www.w3.org/TR/rdf-schema#Class"/>
       <daml:disjointUnionOf rdf:parseType="daml:List">
         <daml:Class rdf:about="#Situation"/>
         <daml:Class rdf:about="#Entity"/></daml:disjointUnionOf>
```

[9] http://www.webkb.org/doc/F_languages.html#FO

```
  <daml:disjointWith resource="#relation"/>
  <daml#sameClassAs resource="http://www.webkb.org/kb/sowa#T"/>
</rdfs:Class>
<rdfs:Class rdf:about="#Thing_playing_some_role">
  <rdfs:subClassOf rdf:about="#Thing"> </rdfs:Class>
```

In FO, the creator of a link is left implicit when it is the same as the creator of the category source of the link. Otherwise, the creator has to be specified within parenthesis (as illustrated above for the name "chose"). To represent link creators in the other notations, either relations with arity higher than two must be used (as illustrated above) or contexts.

"SubtypeOf" links are special cases of *definition of necessary conditions for* (being an instance of) the source categories. Here is an example of how more general cases for the definition of necessary conditions can be represented.

```
E:    According to "pm", by definition, a person has for parent a person.
FCG:  [type pm#person (*x) :=> [*x, pm#parent: a pm#person] ]
KIF:  (defrelation pm#person(?p):=>(exists((?p2 pm#person))(pm#parent ?p ?p2)))
RDF:  <rdfs:Class rdf:ID="Person">
        <rdfs:subClassOf><daml:Restriction>
                           <daml:onProperty resource="#parent"/>
                           <daml:toClass resource="#Person"/>
                         </daml:Restriction></rdfs:subClassOf></rdfs:Class>
```

For definitions of sufficient conditions, `:<=` may be used instead of `:=>`. In RDF/XML, the `rdfs:subClassOf` links must be reversed. For definitions of necessary and sufficient conditions, `EQ`, `:=` and `daml#sameClassAs` may be used.

Function definitions do not exist in RDF. We use `forall` constructs instead, as suggested by Berners Lee [7].

```
E:    The length of a list is 0 if the list is empty,
      otherwise, 1 + the length of the list without its first element
FCG:  [function length (list *l) :-> natural *r
       := [if [l = nil] then [*r = 0] else [*r = 1 + length(rest(*l))] ]]
KIF:  (deffunction length (?l)
        := (if (= ?l nil) 0  (if (list ?l) (1+ (length (rest ?l))))))
RDF:  <forall var="l"><exists var="n">
        <if><daml:list about="#l">
              <daml:sameIndividualAs
                 resource="http://www.daml.org/2001/03/daml+oil.daml#nil"/>
            </daml:list>
         <then><rdf:Description about="#n">
                 <daml:sameIndividualAs>0</daml:sameIndividualAs>
               </rdf:Description></then>
         <else><exists var="l2" v2="n2">
                 <daml:list about="#l">
                   <rest><daml:list about="#l2">
                           <length><natural about="#n2"><plus1 resource="#n"/>
                                   </natural></length></daml:list>
                   </rest></daml:list></exists></else></if></exists></forall>
```

12 Conclusion

We have shown how RDF/XML can be (extented to be) used in various knowledge representation cases and we have proposed intuitive notations (FE, FCG and FO) covering at least all the presented cases. Although these high-level notations are unlikely to be widely adopted, they show some ways to improve other notations in readability, expressivity and "knowledge normalizing effect" – for example, Notation-3, Tim Berners-Lee's "academic exercise", which does not (yet) have have a special syntax for extended quantifiers, collections, functions and definitions.

This article complements the lexical and ontological conventions proposed in [9] to permit knowledge sharing. We are now working on the import and export of FE, FCG, KIF and RDF/XML in WebKB-2, along the lines presented in this article. More information can be found, and testing can be done, on the WebKB site (www.webkb.org).

References

1. The CG specification. http://users.bestweb.net/~sowa/cg/cgstand.htm
2. The KIF specification. http://logic.stanford.edu/kif/dpans.html
   See also: http://www-ksl.stanford.edu/knowledge-sharing/kif/
3. The Knowledge Machine specification. http://www.cs.utexas.edu/users/mfkb/km.html
4. The RDF specification. http://www.w3.org/TR/REC-rdf-syntax/
5. The RDF Schema (RDFS). http://www.w3.org/TR/PR-rdf-schema/
6. The DAML+OIL Schema. http://www.daml.org/2001/03/daml+oil.daml
7. Berners-Lee, T.: The Semantic Toolbox: Building Semantics on top of XML-RDF. http://www.w3.org/DesignIssues/Toolbox.html
8. Martin, Ph., Eklund P.: Knowledge Indexation and Retrieval and the Word Wide Web. In IEEE Intelligent Systems, special issue "Knowledge Management and Knowledge Distribution over the Internet", May/June 2000. http://www.webkb.org/doc/papers/www8/www8.ps
9. Martin, Ph.: Conventions and Notations for Knowledge Representation and Retrieval. In Proc. of ICCS 2000, 8th International Conference on Conceptual Structures (Springer Verlag, LNAI 1867, pp. 41-54), Darmstadt, Germany, August 14-18, 2000. http://www.webkb.org/doc/papers/iccs00/iccs00.pdf
   See also the FE and FCG grammars at http://www.webkb.org/doc/F_languages.html
10. Martin, Ph., Eklund P.: Large-scale cooperatively-built heterogeneous KBs. In Proc. of ICCS 2001, 9th International Conference on Conceptual Structures (Springer Verlag, LNAI 2120, pp. 231-244), Stanford University, California, USA, July 30th to August 3rd, 2001. http://www.webkb.org/doc/papers/iccs01/
11. Sowa, J.F.: Conceptual Graphs Summary. In: Conceptual Structures: Current Research and Practice (Eds: T.E. Nagle, J.A. Nagle, L.L. Gerholz and P.W. Eklund), Ellis Horwood (1992), pp. 3-51.