RDF M&S revisited: From Reification to Nesting, from Containers to Lists, from Dialect to pure XML

Wolfram Conen[+], Reinhold Klapsing[++], and Eckhart Köppen[+++]

[+]XONAR GmbH,
Wodanstr. 7
D-42555 Velbert, Germany,
Conen@gmx.de

[++]Information Systems and Software Techniques,
University of Essen, Universitätsstraße 9,
D-45141 Essen, Germany,
Reinhold.Klapsing@uni-essen.de

[+++]40Hz.org
343A Nature Drive
San Jose, CA 95123
eck@40hz.org

Abstract. The basic result presented is the following: with two (hopefully reasonable) assumptions about the intentions behind the RDF model, it can be shown that the RDF model and a model allowing for nested triple and lists of resources and triples, can be mapped to one another (in both directions). This allows to establish a close link between the RDF model and extended models recently suggested (SlimRDF [3], XRDF [4]). Further, the approach may help to clarify some problems related to interpreting the roles of reification and containers in the RDF model.

1 Introduction

As RDF is considered to be a key ingredient of the evolving semantic web, lack of clarity should be avoided. *Reification* and *Containers* gave rise to a number of discussions. In this paper, we propose an interpretation of these two constructs that may help to clarify this issue. It also demonstrates, how complex expressions can be constructed from RDF that allow a straightforward representation of the modeler's intentions. The basic idea is as follows: In RDF, if someone wants to express a relation between a statement and a resource or two statements, she has to utilize reification. If a relation between an entity (be it a resource or a triple or another group of entities) and a group of entities should be expressed, rdf:Bag, rdf:Seq or rdf:Alt have to be used. Essentially, both constructs are needed to allow expressing nested or grouped constructs with flat triples. Both constructs are not properly tied into the RDF model, for example, the meaning of attaching a predicate to a reificant[1] is not fixed in the model (if r reifies $[s, p, o]$ and $[r, p_2, o_2]$ is given, is the intention to express $[s, p, o]$ *is* p_2*-related to* o_2 or is the intention to take the triple literally, that is r *is* p_2*-related to* o_2?). Fig. 1 and Fig. 2 demonstrate the interpretation of a collection of flat-triple statements as one nested triple.

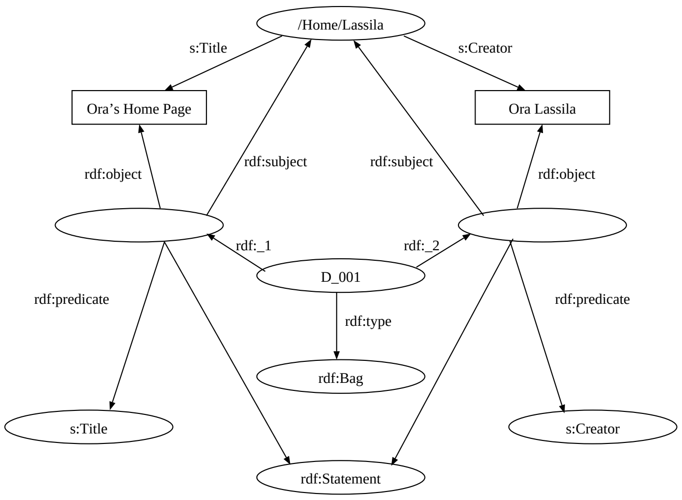

Figure 1: Demonstrating bags and reification (Figure 9 in [2])

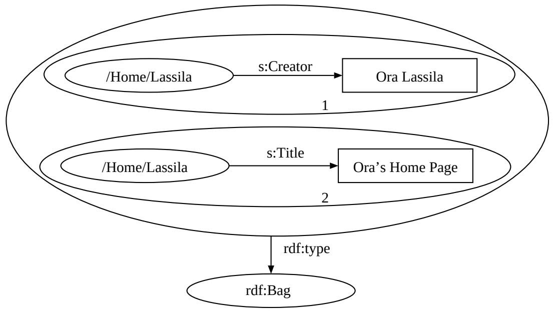

Figure 2: An intention-equivalent nested representation.

We decided to fix the possible interpretation of the flat-triple constructs *reification* and *containers* by assuming that each reification is only a surrogate for the triple it represents and each container is only a surrogate for a list of entities[2], that is, in a natural representation of the intentions, each reification and each container will be substituted by the represented triple or list and all only technically necessary triples of the flat model will be eliminated. We will argue that such a non-flat model captures the *essence* of the initial set of statements. In the following, the two underlying assumptions will be presented more precisely.

1.1 Assumptions

Let S be a set of flat-triple statements.

[1]This terminology will be explained shortly.
[2]An entity may be a resource, a literal, a statement or a list of entities.

Assumption 1: Be r the *reificant*[3] of the triple $[s, p, o]$ as defined in [2], that is, the set S contains the triples $[r, \text{rdf:subject}, s]$, $[r, \text{rdf:predicate}, p]$, $[r, \text{rdf:object}, o]$ and $[r, \text{rdf:type}, \text{rdf:Statement}]$ (we will generally present triples in an infix[4] sequence, that is as $[subject, predicate, object]$). The reification is used in another statement, say $[s_1, p_1, r]$. *Now, we assume that the intentions behind this subset of statements is to express that p_1 relates s_1 to $[s, p, o]$. This intention can easily be captured in a model allowing nested triples as $[s_1, p_1, [s, p, o]]$.*

Assumption 2: Be c a container, for example of type Seq, that is $[c, \text{rdf:type}, \text{rdf:Seq}] \in S$. The n-ary sequence $R_C = (r_1, \ldots, r_n)$ of resources contains the elements of c, that is, $[r, \text{rdf:\_}i, r_i] \in S$ for all $i, 1 \leq i \leq n$. The container is used in another element of S, say $[s, p, c]$. *Now, we assume that the intentions behind this subset of statements is to express that p relates s to $(r_1, \ldots, r_n)$. This intention can easily be captured in a model allowing for lists of resources as $[s, p, (r_1, \ldots, r_n)]$.*[5]

Based on these two assumptions, the role of containers and reification can intuitively be described as allowing to express structure of arbitrary complexity with the limited instrument of a model based on flat triples.

In the next section, an extended model will be suggested that directly captures the underlying intentions of the constructs *reification* and *container* by introducing nesting and lists. In Section 3, we will prove that every RDF model can be expressed as an extended model *and* that every extended model can be expressed as an RDF model. This may suggest that the more comprehensive representation of the RDF model (that is: the extended model) may be prefered when RDF models are used in applications. In Section 4 this aspect is briefly explored and two representation of the extended model, namely a graph notation and a straightforward XML DTD are suggested. In Section 5, two issue that explore the relation between structural and semantical aspects may give hints on possible directions for developing semantics for the extended models. Section 6 concludes the paper with a brief discussion.

2 An Extended Model

Let A be a reasonably selected alphabet. Let A^* be the set of strings defined over A. The following grammar defines expressions over A^* of the form R recursively as

$$R \quad ::= \quad r \quad | \quad '(' \, R \, ',' \, R \, ')' \quad | \quad '[' \, R \, ',' \, R \, ',' \, R \, ']'$$

Here, r denote elements of A^*. (Sub-)Expressions of this form will be called *atomic*. A (sub-)expression of form R is called *resource*. A (sub-)expression of form R which matches the pattern $[R, R, R]$ is called *statement*. A (sub-)expression of form R which matches the pattern (R, R) is called *list*. Note that we will frequently use $(r_1, r_2, \ldots, r_n)$ instead of the more cumbersome $(r_1, (r_2, (\ldots, (r_{n-1}, r_n) \ldots)$ where this can be done without the risk of misinterpretation. We will also leave out the comma regularly. Furthermore, we will only consider

[3]In [2], this is called *reified statement*, which might be a bit confusing–there is something that is reified, yes, but that is the "original" statement $[s, p, o]$. Instead of *reificant*, *reifying resource* could be used. Note, that it would not be very useful to say *reifying statement* because r is **not** defined to be a member of the set **statement** (which is a concept of the RDF model defined in [2]), instead r has the **type** *rdf:Statement*, which is, for the core RDF model, only a string representing a resource, and requires an interpretation in the context of RDF Schema and its constraints and concepts.

[4]infix with respect to the predicate.

[5]Note, that both assumptions together can be used to build list of statements etc.

finite sets of finite expressions.

3 Relation between Extended model and RDF model

Let us begin with a remark: we will assume that the sets of statements of the RDF model that we will consider below are, in a certain sense, well-behaved, that is, we will assume that no reificant nor container is part of the (possibly complex) structure that it represents[6]. We will capture this more precisely in an algorithm that tries to order resources in strata so that each resource in a stratum represents structures that consist of resources of lower ranking strata. The algorithm can also be used to detect whether its input (a set of statements of the RDF model), is not well-behaved.

Below, we will show that extended model and RDF model can be mapped to another. First, some definitions are needed to prepare the stage. In the following, $s, p, o, s_1, o_1, r_1, \ldots, r_n$ will all be entities, that is, either an atom, a statement or a list if the extended model is considered or resources or literals (only possible in object position) if the RDF model is considered.

Note that the definitions 1 and 3 can be applied to both models.

Definition 1 (Reification).
Given a resource r and the following set of statements, T^r:

$$T^r = \{ \quad [r, \textit{rdf:subject}, s], [r, \textit{rdf:predicate}, p], [r, \textit{rdf:object}, o], [r, \textit{rdf:type}, \textit{rdf:Statement}] \quad \}$$

Then, r is called a reificant *of $[s, p, o]$ and T^r is called a* reification *of $[s, p, o]$.*

Definition 2 (Reification: Derivation, Consequence).
Let u be a nested statement of the extended model of the form $[[s, p, o], p_1, o_1]$. Let r be a reificant of $[s, p, o]$ and T^r be the corresponding reification. The set $D = [r, p_1, o_1] \cup T^r$ is called a derivation *of u. u, in turn, is called a* consequence *of D (analogously defined for $u = [s_1, [s, p, o], o_1]$ and $[s_1, p_1, [s, p, o]]$). With respect to a set $\mathcal{C}$ of statements, we say that u is* derivable *in $\mathcal{C}$ if $D \subseteq \mathcal{C}$.*

Definition 3 (Container).
Given a resource c and the following set of statements (with $X \in \{$rdf:Seq, rdf:Alt,rdf:Bag $\}$): $T^c = \{ [c, \textit{rdf:type}, X]\} \cup \{[c, \textit{rdf:\_i}, r_i] \mid i \in \mathbb{N}, 1 \leq i \leq n\}$. Then, c is called a container, *T^c is called an n-ary* container definition *and the n-ary sequence $R_c = (r_1, \ldots, r_n)$ of entities is called the* elements *of c.*

Definition 4 (Container: Derivation, Consequence).
Let u be a nested statement of the extended model of the form $[(r_1, \ldots, r_n), p, o]$. Let c be a container for the elements $(r_1, \ldots, r_n)$ of the Seq-type and let T^c be the corresponding container definition. The set $D = [c, p, o] \cup T^c$ is called a derivation *of u. u, in turn, is called a* consequence *of D (analogously defined for $u = [s, (r_1, r_2, \ldots, r_n), o]$ and $[s, p, (r_1, r_2, \ldots, r_n)]$). With respect to a set $\mathcal{C}$ of statements, we say that u is* derivable *in $\mathcal{C}$ if $D \subseteq \mathcal{C}$.*

[6]In the RDF model, as it is described in [2], it is, for example, possible to reify a statement that contains the representing reificant–which should, almost certainly, not be allowed. The same goes for containers containing themself.

Note that a *consequence* can not be a statement from the flat RDF model. However, the *derivation* of a statement with only one level of nesting can completely consist of statements from the flat model. To be able to define some notion of equivalence between sets of statements from the extended and the flat model, we have to define how a deeply nested statement can be derived recursively from a set of flat statements.

Definition 5 (Rooted, Root, Hull). *Be $\mathcal{O}$ a set of flat statements from the RDF model. Let $\mathcal{N}$ be a set of statements from the extended model. Let u be a statement from the extended model. We say that u is* rooted *in $\mathcal{O}$ if either*
(1) $u \in \mathcal{O}$ or
(2) a derivation D of u can be given such that each statement $t \in D$ is rooted *in $\mathcal{O}$.*
We say that $\mathcal{N}$ is rooted *in $\mathcal{O}$ if every statement n of $\mathcal{N}$ is* rooted *in $\mathcal{O}$. $\mathcal{O}$ is called* root *of u if u is rooted in $\mathcal{O}$; it is called* minimal root *of u if it is a root of u and for any statement $t \in \mathcal{O}$, u is not rooted in $\mathcal{O}\backslash\{t\}$. $\mathcal{O}$ is called* root *of $\mathcal{N}$ if every statement $n \in \mathcal{N}$ is rooted in $\mathcal{O}$; it is called* minimal root *of $\mathcal{N}$ if it is a root of $\mathcal{N}$ and for any statement $t \in \mathcal{O}$ a statement $n \in \mathcal{N}$ can be found such that n is not rooted in $\mathcal{O}\backslash\{t\}$. We say that $\mathcal{N}$ is a* hull *of $\mathcal{O}$, if $\mathcal{O}$ is a minimal root of $\mathcal{N}$ – we will alternatively say that $\mathcal{N}$ and $\mathcal{O}$ are* intention-consistent *(or simply* consistent*).*

Example The following set $\mathcal{O}$ of flat statements is a *minimal root* (that is, no statement can be removed from $\mathcal{O}$) for the nested statement [Gustaf says [Ecki likes (Reinhold Wolfram)]]: { [Gustaf says r_1], [r_1 rdf:type rdf:Statement], [r_1 rdf:subject Ecki], [r_1 rdf:predicate likes] [r_1 rdf:object l_1], [l_1 rdf:type rdf:Seq], [l_1 rdf:_1 Reinhold], [l_1 rdf:_2 Wolfram] }. The sets { [Gustaf says [Ecki likes (Reinhold Wolfram)]] } or { [Gustaf says [Ecki likes (Reinhold Wolfram)]], [Gustaf says r_1], [r_1 rdf:predicate likes} are, among finitely many others[7], are *hulls* of $\mathcal{O}$.

We look for hulls that contain only the minimally necessary number of statements to capture, with respect to the above asumptions, the intentions of the underlying set of flat statements.

Definition 6 (Essence). *Be $\mathcal{O}$ a set of flat statements from the RDF model. Let $N^{\mathcal{O}} = \{\mathcal{N}|\mathcal{N}$ is a set of statements from the extended model $\wedge$ $\mathcal{N}$ is a hull of $\mathcal{O}\}$ be the* set of possible hulls *of $\mathcal{O}$. The subset of $N^{\mathcal{O}}_{\min} = \{\mathcal{N} \in N^{\mathcal{O}} \mid \nexists \mathcal{M} \in N^{\mathcal{O}}$ with $|\mathcal{M}| < |\mathcal{N}|\}$ is the* set of minimal hulls. *An element of $N^{\mathcal{O}}_{\min}$ is called a* minimal hull *or* essence *of $\mathcal{O}$ – we will alternatively say that $\mathcal{N}$ and $\mathcal{O}$ are* intention-equivalent *(or simply* equivalent*).*

Note that due to the definitions of derivations, the minimal hull of a given set of statements from the RDF model is unique.

Example The minimal hull for the set $\mathcal{O}$ of the above example is { [Gustaf says [Ecki likes (Reinhold Wolfram)]] }.

Now, the following two propositions can be proved. The first one essentially states, that each set of extended statements can be expressed as an intention-consistent set of flat statements, while the second will show that each set of flat statements can be expressed as an intention-equivalent set of extended statements.

Proposition 1: *For each set $\mathcal{N}$ of statements from the extended model, a set $\mathcal{O}$ of statements from the RDF model can be found such that $\mathcal{O}$ is a minimal root of $\mathcal{N}$.*

[7]In contrast, for a set of nested statements there is usually an infinite set of possible minimal roots due to the possible variations in naming the necessary containers and reificants.

Proof: Intuitively, each nested statement can be expressed with a set of flat statements that allows to derive, possibly incurring intermediate nested statements, the initial statement (some care has to be taken not to confuse the *symbols* used in the model). Let us consider an example:

Initial Expression:	[Gustaf says [Ecki likes (Reinhold Wolfram)]]
First Step:	[Gustaf says [Ecki likes l_1]]
(add derivation of	[l_1 rdf:type rdf:Seq]
the list)	[l_1 rdf:_1 Reinhold], [l_1 rdf:_2 Wolfram]
Second Step:	[Gustaf says r_1]
(add derivation of	[r_1 rdf:type rdf:Statement], [r_1 rdf:subject Ecki]
the embedded	[r_1 rdf:predicate likes], [r_1 rdf:object l_1]
statement)	[l_1 rdf:type rdf:Seq]
	[l_1 rdf:_1 Reinhold], [l_1 rdf:_2 Wolfram]

This can be formalized as follows. Be C^E a set of statements[8] from the extended model. With the following construction, an intention-consistent set C^R of statements from the RDF model can be determined.

```
Algorithm Flaten(In: $C^E$)
(1) $C^R = \emptyset$. Foreach $t \in C^E$ do
(2)     Expand($t$,0,$C^R$)

(1) Function Expand (In: Expression $t$, In: Int $l$, InOut: Set of Statements $E$) returns a Symbol
(2)         If $t \in A^*$ then return $t$
(3)         If $Form(t)$ =Statement (matching $[s, p, o]$) then
(4)           $s_r$ = Expand(s,l+1,E); $s_p$ =Expand(p,l+1,E); $s_o$ = Expand(o,l+1,E)
(5)           $r$ =Symbol[9](t);
(6)           if $(l = 0)$[10] then $E = E \cup \{ [s_r, s_p, s_o]\}$; return EmptySymbol else
                $E = E \cup \{ [r,$ rdf:type, rdf:Statement$]$,
                $[r,$ rdf:subject$, s_r]$, $[r,$ rdf:predicate$, s_p]$ $[r,$ rdf:object$, s_o]$ $\}$; return $r$
(7)         If $Form(t)$ =List (matching $(r_1, \ldots, r_n)$) then
(8)           $r$ =Symbol(t);
(9)           $E = E \cup \{ [r,$ rdf:type, rdf:Seq$]\}$
(9)           For $1 \leq i \leq n$ do
(10)              $s_i$ =Expand($r_i$,l+1,E)
(11)              $E = E \cup \{ [r,$ rdf:_$i, s_i] \}$
(12)          return $r$;
```

Let us sketch the proof of the correctness of the algorithm: (1) The algorithm terminates. To see this, consider the following: The function *Expand* recursively descents through the structure of its input expression. It will stop the descent in each branch of the structure as soon as

[8]Arbitrary sets of expressions resp. resources could also be allowed. This would require only a simple, but unnecessary (for this presentation) extension.

[9]The function $Symbol$ returns a new symbol for each subexpression t that is not already represented in the flat model, otherwise, the already known symbol will be returned. This will be discussed below

[10]The top-level expression is always a statement. There is no need to reify this statement because the reificant would be left unused (in this particular expansion).

an element of A^* is found (which will ultimately be the case, as the expressions have been assumed to be finite). Furthermore, each subexpression branch will be considered exactly once. (2) C^E and and the computed set C^R are intention-consistent. To see this, consider the following. First note that all statements added to C^R are flat. The algorithm constructs a derivation for each top-level statement by constructing derivations for each embedded expression while returning from the descent. Thus, each statements of C^E is rooted in the constructed C^R. C^R is a minimal root because no other statements but elements of derivations are added to C^R. $\boxminus$

A note regarding the function *Symbol*. In the algorithm, we have chosen to compute one *unique* name for literally identic subexpressions, that is if, say, the statement [Ecki likes RDF] is encountered twice, it will be reified only once (although, to keep the algorithm above simple, it will be flatened twice but this will result in an identic set of flat statements and the redundancy will thus vanish due to considering sets). This makes it easy to identify literally equivalent expressions in the flat model (they have the same "name"), it, however, may make it more difficult to explore the differences in the meaning of multiple occurences of literally identic expressions (this will have to consider the structural context of such expressions – sensibly dealing with this kind of context should be made possible in the semantics build upon this models, so, for a full discussion of the implications, precise objectives for semantics are required. It is, nevertheless, easy to generate a new symbol for each occurence of literally identic subexpressions, if this is found to be the better way to go).

Proposition 2: *For each set $\mathcal{O}$ of statements from the RDF model, a set $\mathcal{N}$ of statements from the extended model can be found such that $\mathcal{N}$ is a minimal hull of $\mathcal{O}$.*

Proof: (constructive) The resources used in the RDF model can be arranged in strata if it is assumed that no circular definitions of reifications resp. containers exist (see below for details). The following algorithm will either determine a stratification or detect that circular references exist.

```
*Algorithm Stratification(In: $C^E$)*
Initially, all resources and literals are unmarked.
*[Compute Stratum 0]* Mark all literals as being in stratum 0. Mark all resources that are
neither a reificant nor a container as being in stratum 0.
*[Compute further Strata]* **while** there is a resource $r$ that is *unmarked* **and** all the resources
or literals that are represented by $r$ (this set of entities will be called $E$)[11] are marked **do**
    Determine the highest marking, say $j$, of a resource in $R$.
    Mark $r$ to be in stratum $j + 1$.
*[Check validity]* **if** an unmarked resource exists
    **then** return "ERROR: there are mutually referencing structures"
    **else** return "OK - a stratification has been determined".
```

With the above assumption of a finite input set and finite expressions, the algorithm eventually terminates (in each round, an unmarked resource is marked). If the algorithm prints out the "OK" message, the following condition will hold: each resource r that represents a structure

[11]If r is a reificant and $[s, p, o]$ is a statement that r reifies then s,p and o are in E. If c is a container then the elements of R_c (as defined above) are in E. Note that with the definition of container above, a set of flat statement that defines a n-ary container also defines $(n-1)$, $(n-2)$ ... -ary containers. We assume that E contains all eintities that are elements of the container with the largest arity. Besides this solution for the case in which r represents more than one structure, all other cases should probably be considered an error (for example, a resource that represents two statements or a statement and a container, or two non-inclusive containers).

belongs to a higher stratum then the resources/literals that are part of the represented structure. If the algorithm returns an ERROR message, at least one resource represents a structure that contains either the resource or a structure that, if recursively dereferenced, contains the resource.[12]

From finiteness follows that a highest ranking non-empty stratum, say k exists. Furthermore, it follows from the construction that the strata that are formed by marking resources and literals as their elements, are consecutively numbered.

Example This is an example of a mapping from a stratified set of flat statements to an extended statement:
Input: $\{$ [Gustaf says r_1] [r_1 rdf:subject Ecki], [r_1 rdf:predicate likes], [r_1 rdf:object l_1], [r_1 rdf:type rdf:Statement], [l_1 rdf:type rdf:Seq], [l_1 rdf:_1 Reinhold], [l_1 rdf:_2 Wolfram] $\}$
Stratum 0: $\{$ Gustaf Reinhold Wolfram Ecki says likes rdf:object rdf:type rdf:subject rdf:predicate rdf:Statement rdf:Seq rdf:_1 rdf:_2 $\}$
Stratum 1: $\{l_1\}$
Stratum 2: $\{r_1\}$
Now a mapping along the stratification can be performed. First, $\{$ [l_1 rdf:type rdf:Seq], [l_1 rdf:_1 Reinhold], [l_1 rdf:_2 Wolfram] $\}$ is mapped to (Reinhold Wolfram), the statements are removed from the initial set, and each occurence of l_1 is replaced by (Reinhold Wolfram), leading to the next set of statements (now already extended): $\{$ [Gustaf says r_1] [r_1 rdf:subject Ecki], [r_1 rdf:predicate likes], [r_1 rdf:object (Reinhold Wolfram)], [r_1 rdf:type rdf:Statement], [(Reinhold Wolfram) rdf:type rdf:Seq] $\}$. Next, the set $\{$ [r_1 rdf:subject Ecki], [r_1 rdf:predicate likes], [r_1 rdf:object (Reinhold Wolfram)], [r_1 rdf:type rdf:Statement] $\}$ is mapped to [Ecki says (Reinhold Wolfram)] which replaces r_1, resulting in the minimal hull, [Gustaf says [Ecki likes (Reinhold Wolfram)].

This is captured in the following algorithm. It will determine an intention-equivalent extended model, C^E from a set of statements of the RDF model, C^R.

```
*Algorithm Nest*(In: $C^R$)
**For** stratum $s = 1$ to $k$ **do**
  **For all** resources $r$ in $s$ **do**
    **if** $r$ is a reificant **then**
      Remove from $C^R$ the four statements defining the reification
        which has $r$ as a reificant.
      Replace all occurences of $r$ in expressions in $C^R$ by the statement that $r$ reifies.
    **if** $r$ is a container **then**
      Remove from $C^R$ all statements of the form $[r, \_i, r_i]$ and
        build a list $R_r$ from the resources $r_i$
      Replace all occurences of $r$ in expressions in $C^R$ by the list $R_r$
$C^E$ = $C^R$.
```
Again, the proposition follows from the construction. $\boxminus$

The relation between the RDF model and the extended model relies on the two assumptions. If these assumptions are not accepted as being a reasonable interpretation of the intentions behind the RDF model, then the propositions and proofs given above do not hold. However, the newly introduced model may still be considered as a reasonable, comprehensive alterna-

[12]Note that this can also be a chain of reifications and containers, that is, we consider it to be an error if a container contains a reificant that reifies a statement that contains the container etc.

tive to the RDF model due to its ability to capture complex expressions naturally. We will try to illustrate this by suggesting two alternative representations in the next section, namely a graphical notation and a XML DTD, which both may be considered as advantageous[13] if compared to the alternatives offered in the RDF M&S specification.

4 Graphical and XML Representations of the Extended Model

The following graphical examples and the XML DTD largely follow the presentation in the XRDF discussion paper [4]. Both offer (reversibly mapable) alternatives to the statement/list notation of the extended model.

4.1 The Extended Model as a Graph

The graphical language for the extended model provides the constructs oval (representing resources) and directed labeled arcs (representing relations). Each oval representing a resource of the atom-type has inscribed a content taken from the alphabet A^*. Each embedded oval will be augmented with a number that is unique within the oval it is directly embedded in. Numbers will be left out where possible (ie., in statements, where the ordinal number follows from the direction of the arc, and in lists with one element only). A precise transformation to and from the nested-triple notation of the extended model is straightforward (compare [4]). Some examples of the graphical notation are given in the figures below.

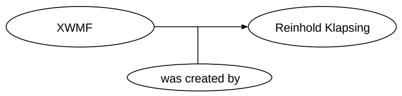

Figure 3: Representing “XWMF was created by Reinhold Klapsing”

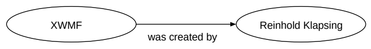

Figure 4: The same statement, now neglecting the fact that the predicate is also a resource (which is the usual way).

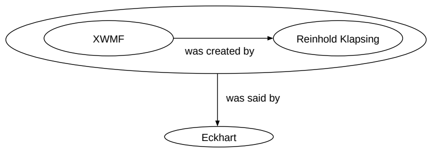

Figure 5: Representing “‘XWMF was created by Reinhold Klapsing’ was said by Eckhart.”, a statement about the previous statement.

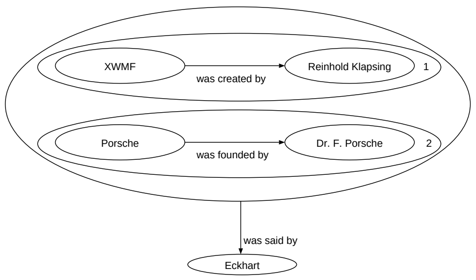

Figure 6: Eckhart made two statements.

It is left to the reader to flaten the graphically represented statement of the extended model to corresponding sets of flat RDF statements. This may suffice to demonstrate that already mildly complex examples of modeling tasks are much more straightforward to formulate (either graphical or in triple notation) with the extended model than with the flat model. Given

[13] Aware: subjective judgement. We hope, however, that some readers may share our opinion.

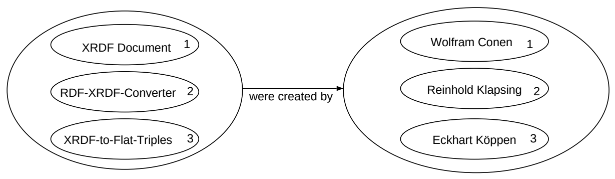

Figure 7: A group of people jointly created a collection of artifacts.

the intention-equivalence of the two models (based on the two assumptions stated above), the extended model seems the more convenient way to express the intentions of set of flat model statement in which reification and containers are used.

4.2 A pure XML syntax for the Extended Model

It is straightforward to represent the (few) ingredients of the nested/list model as an XML-DTD (compare [4] with slightly different list and predicate definitions).

```
<!ELEMENT statement (subject, predicate, object)>
<!ELEMENT list      (statement | atom | list)+>
<!ELEMENT atom      (#PCDATA)>
<!ELEMENT subject   (atom|statement|list)>
<!ELEMENT predicate (atom|statement|list)>
<!ELEMENT object    (atom|statement|list)>
```

A conversion of an XML document that conforms to the above DTD into an extended model is immediate. The algorithm *Flaten* from above gives a conversion to an RDF model. From this, a XML/RDF representation (at least a direct, explicit representation where each statement results in one description) can be derived easily.

Example: The statement [(XWMF was_created_by Reinhold_Klapsing) (Porsche was_founded_by Dr._F._Porsche) was_said_by Eckhart] , compare Figures 6 above, can be "serialized" as follows:

```
<statement>
  <subject><atom>Eckhart</atom></subject>
  <predicate><atom>says</atom></predicate>
  <object>
    <list>
      <statement>
        <subject><atom>XWMF</atom></subject>
        <predicate><atom>was_created_by</atom></predicate>
        <object><atom>Reinhold Klapsing</atom></object>
      </statement>
      <statement>
        <subject><atom>Porsche</atom></subject>
        <predicate><atom>was_founded_by</atom></predicate>
        <object><atom>Dr. F. Porsche</atom></object>
      </statement>
    </list>
  </object>
</statement>
```

There is a number of reasons that make pure XML an attractive alternative to the RDF/XML serialization dialect, we refer the interested reader to the XRDF discussion paper for further details. We will now briefly turn our attention to basic semantic aspects that are closely related to the nested structure of expression of the extended model.

5 Some brief considerations of Semantics

To facilitate the authoring and deployment of meta-data, syntactical and structural simplicity is needed. A layered approach has proven to be useful for the definition of models and techniques (this is especially obvious in the context of XML-based standards, where XML is the basis for other standards like namespaces which in turn are used in the definition of XSLT).With the extended model proposed above, we define the syntax of a lowest layer, make use of the structural primitives statement and list. On top of this basic structural model, semantic definitions and interpretations can be layered. Though this is not the main topic of the paper, we will briefly discuss two aspects that are related to semantical explorations of nested structures.

5.1 Exploring/Propagating Meaning from Outside to Inside

For the task of designing suitable semantics with the extended model we will have to consider a number of design options. We will propose one possible route and point out a few more things that might come in handy. Our route makes use of the following key observation: the semantics related to (sub-)expressions depend on their position within the surrounding structure – that is, the semantics will be explored starting from the outermost part of the structure and proceeding to the innermost part. Let's consider an example that demonstrates a simple kind of truth predicate.

[[sky color blue] hasTruthValue FALSE]]

or, in a flatened version

[r type statement][r subject sky][r predicate color][r object blue] [r hasTruthValue FALSE]

The following transformation and constraints will give the flatened version some meaning in a FOL representation:

Transformation: Map each triple [s,p,o] into an instance of a predicate triple(s,p,o).

Constraints :

(1) reifies(R,S,P,O) $\leftarrow$
    triple(R,type,statement) $\wedge$ triple(R,subject,S) $\wedge$
    triple(R,predicate,P) $\wedge$ triple(R,object,O).
(2) falsified_resource(R) $\leftarrow$
    statement_known_as_true(R,hasTruthValue,FALSE).
(3) statement_known_as_false(S,P,O) $\leftarrow$
    triple(S,P,O) $\wedge$ reifies(R,S,P,O) $\wedge$ falsified_resource(R).
(4) statement_known_as_true(S,P,O) $\leftarrow$
    triple(S,P,O) $\wedge$ not(statement_known_as_false(S,P,O)).

The statement_ . . . -predicates could be used for further inferences. Note that further embedding works out fine also, ie. falsifying falsified statements is possible. The key point here is that the truth of the information contained in a triple will be propagated from the outermost expression to the innermost. This principle can be used to define more sophisticated semantics as well, as would be necessary to give a proper meaning to expressions like

[Reinhold believes [Ecki assumes [[Wolfram is nice] hasTruthValue FALSE]]]

It is clear that the actual meaning of embedded expressions depends on the "semantical" scope that is propagated from the outer predicates.

```
Scope 3: (believes  Reinhold
Scope 2:      (assumes  Ecki
Scope 1:            (hasTruthValue FALSE
Scope 0:                   (is Wolfram nice))))
```

What is actually *done* with this information will depend on the proper definition of semantics for predicates and their interaction. With respect to the above example the following question should be answered: what should be the meaning of an elementary statement of which someone believes that someone else assumes that its negation is true. Here, an "elementary" statement can be defined as a statement that has a predicate that does not modulate the truth value of the subject or object (like *is* in the above example). This may suffice to show how the meaning of statements generally depend on their *position*. From the intention-equivalence of extended and RDF model shown above, it follows that this is also true for RDF models – note, that the "position" of a statement in a complex structure is given by its occurence in reifications/lists, for example, the following set of statements flatens the above expression:

[Reinhold believes r_3]
[r_3 subject Ecki] [r_3 predicate assumes] [r_3 object r_2] [r_3 type Statement]
[r_2 subject r_1] [r_2 predicate hasTruthValue] [r_2 object FALSE] [r_2 type Statement]
[r_1 subject Wolfram] [r_1 predicate is] [r_1 object nice] [r_1 type Statement]

Note that there is no need (or better: no use) to "materialize" the intermediate "propositions" like [Ecki assumes r_2], for, if this would be done, an intention-equivalent extended model would contain two statements:

[Reinhold believes [Ecki assumes [[Wolfram is nice] hasTruthValue FALSE]]]
[Ecki assumes [[Wolfram is nice] hasTruthValue FALSE]]

which is somewhat different from having only the first statement, because now, [Ecki . . .] has become a factual statement. It should also be clear from this example that, within the scope of different statements, literally identic subexpressions can have different meaning. This also demonstrates that it is not necessary to give every occurence of literally identic subexpressions an unique identity, because the actual meaning of each occurence depends on its context, which is captured by the position of the subexpression within other expressions.

5.2 Abbreviating Expressions with Structural Transformations

It is possible to provide some kind of syntactic sugar with the help of structural transformations that map a "sugarized" notation to the regular extended model. Some possible transformations are discussed in [4]. This creates the possibility to specify for a predicate which type of transformation should be performed prior to interpreting the predicate. The transformations can also be applied recursively and can make use of indirection (see example below). This touches upon basic layers of semantics for which an extensive discussion is beyond the scope of this paper. We will therefore only give two brief examples.

The predicate *likes* is defined to be of the transformation type $n \times m$, that is a statement of the form [$(n_1, \ldots, n_k)$ likes $(m_1, \ldots, m_l)$] will be *expanded* to the list of statements $([n_1$ likes $m_1] \ldots [n_1$ likes $m_l]$ $[n_2$ likes $m_1] \ldots [n_k$ likes $m_l])$. So,

[Wolli likes (Reinhold Eckhart)]

is transformed to

([Wolli likes Reinhold] [Wolli likes Eckhart])

Further assume that a specific predicate, *representedBy*, can be used to give *names* to lists (a similar predicate will exist for statements), like in

[(Reinhold, Eckhart) representedBy Friends]

Now, the type of the predicate*likes* can be adapted to the possibilities of *indirection*, that is if a *name* is encountered in subject or object position, the predicate will not be applied to the name (which is a resource itself) but to resource or list of resources that is represented by that name. We will denote the transformation type of *likes* accordingly as $in \times im$. Now, the following becomes possible:

[Wolli likes Friends]

which will result in the same list of statements as above. Note that this or similar kinds of indirection can be used to cleanly seperate between relations to a resource and to the resources (lists/statements) that may be represented by resources.

This technique of basic transformation that may be applied prior to assessing the complete semantics of predicates, may easily be used to answer the above question:

Assume that *describedBy* is a predicate of the transformation type *descriptive* which takes a list with an even number of elements in object position as an input to the transformation which performs the following:

[Reinhold describedBy (hasName Klapsing hasAddress Essen)]

which will be transformed to

([Reinhold hasName Klapsing] [Reinhold hasAdress Essen])

Whether this kind of transformations should be part of a basic layer of (pre-)semantics certainly remains to be discussed.

6 Discussion

Let us briefly discuss one of the potential problems of upgrading from the RDF model to the extended model: the typing of containers. The issue is that two containers with definitions that refer to the same sequence of elements but with different types (e.g., one Bag, one Seq) become indistinguishable with the above mapping into the extended model. Allow two brief remarks: (1) one solution is to relegate this kind of typing of containers to a schema level.

Together with, for example, the *representedBy* property described in 5.2, names for lists can be introduce and types for the lists represented by the names can be attached to the names etc. (2) Another solution is to drop the typing of containers and to simply regard them all as sequences – and to attach the information how a lists should be treated to the properties that make use of the list (each property can interpret a sequence as a Bag or an Alt construct if this suits the definition of the semantics of this property). More alternatives exist and a solution (an adaptation of the mapping) should be provided when defining a schema level for the extended model

Certainly, more details could be explored and more questions should be asked and answered[14]. However, this may suffice to demonstrate that the nesting of statements and the use of list of statements and resources may allow for a natural representation of useful structures that are cumbersome to model and difficult to use in RDF. Based on the interpretation of reification and containers given above, the RDF model (or, intention-equivalently, the extended model) can be seen as providing a (relatively rich) abstract syntax to build rather complex expressions. This may ease modeling with RDF (respectively with the extended model) and may also provide a more clear-cut syntactic layer for the schema layer(s) to be put on top of this model.

References

[1] Dan Brickley and R.V. Guha. Resource Description Framework (RDF) Schema Specification 1.0. Candidate Recommendation, W3C, March 2000. http://www.w3.org/TR/2000/CR-rdf-schema-20000327.

[2] Ora Lassila and Ralph R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. Recommendation, W3C, February 1999. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222.

[3] Sergey Melnik. Slim RDF. Email to the RDF-IG, 12. February 2001. http://lists.w3.org/Archives/Public/www-rdf-interest/2001Feb/0090.html

[4] W. Conen, R. Klapsing and E. Koeppen. XRDF - an eXtensible Resouce Description Framework Discussion Paper, November 2000. http://nestroy.wi-inf.uni-essen.de/rdf/xrdf/

[14]Don't hesitate to contact us if you found your questions unanswered, want to contribute ideas, or simply want to share your opinion on RDF/extended models with us. We thank Graham Klyne and an anonymous reviewer for comments on a previous version of the paper