# Revisiting and Simplifying RDF

Benjamin Heitmann[1], Eyal Oren[1], and Max Völkel[2]

[1] Digital Enterprise Research Institute
National University of Ireland, Galway
Galway, Ireland
first.last@deri.org
[2] FZI Forschungszentrum Informatik Karlsruhe
voelkel@fzi.de

**Abstract** RDF, a simple yet expressive data model, is widely recognised as the foundation for the Semantic Web. We revisit the RDF specification and analyse five problems: literals are not addressable, blank nodes are not addressable, reification is not semantically recognised, language tags are not addressable, and language tags can not be combined. We introduce SiRDF, a simplified data model with fewer modelling elements that overcomes the mentioned problems, and formally show its semantic equivalence to RDF. Based on the lessons of SiRDF we recommend a restricted usage of RDF that solves the mentioned problems while fully adhering to the original RDF specification.

## 1 Introduction

The Semantic Web has evolved, as depicted in the Semantic Web layer cake[3]. Attention shifted away from lower layers, such as XML and RDF, to higher layers, such as the FOAF[4] and SIOC[5]) vocabularies or the RDF Schema [1], OWL [15] and WSML [2]) ontologies. Still, some modelling issues remain in the basis of RDF, leading to problems that are not addressed in higher layers.

In this paper we look closely at RDF and identify five problems: (a) the occurrence of literals is not addressable, (b) blank nodes are not addressable, (c) reification is not recognised by the formal semantics. (d) language tags are not addressable, and (e) language tags cannot be combined. Although these limitations are by design, we show their impact on real world use cases, and therefore propose how to address and resolve them.

We discuss these problems in detail in Sect. 2 and provide references for the real-world impact of these modelling issues. In Sect. 3, we introduce a simplified version of RDF (SiRDF). SiRDF has a smaller data model in terms of modelling constructs. We proof that SiRDF can correctly represent all RDF statements and show that SiRDF addresses the discussed problems.

[3] http://www.w3.org/DesignIssues/diagrams/sweb-stack/2006a.png
[4] http://www.foaf-project.org
[5] http://sioc-project.org/

Proposing SiRDF as a new foundational language for the Semantic Web has limited practical value. In Sect. 4 we therefore map the SiRDF data model back to RDF and get "restricted RDF", a subset of RDF. We show how restricted RDF solves the aforementioned problems to a large extent.

Some recommendations proposed in Sect. 4 have been suggested before in best-practices guides, in weblog posts, or in implementation documentation. The contribution of this paper lies in a thorough analysis of these problems and in a formally underpinned solution.

## 2 Issues with RDF

RDF is a language intended for expressing propositions using precise formal vocabularies [10]. The data model of RDF is a collection of triples (subject, predicate and object) that together form a graph. The elements of the graph are either URI references, literals or blank nodes.

URIs have an identifier and can thus be addressed from outside the graph. Blank nodes are unnamed and can only be addressed from within the graph. Literals represent values by their lexical representation. Literals can be annotated with a language tag or data type; these annotations are not in the form of statements, but rather attachments to the literal. URI references are allowed anywhere in the triple, blank nodes are allowed as subjects and objects, and literals can only be objects.

The simplicity of RDF was an explicit design goal [12, Sect. 2.2], but as we explain in this section, the details of the data model and formal semantics lead to problems that impact real-world use cases. In the remainder of this section, we identify these issues and illustrate each problem with a concrete example in RDF from a simple scenario: two persons, Peter and Paula, document information about themselves in RDF, using the Friend of a Friend (FOAF) vocabulary.

### 2.1 Literal occurrences are not addressable

We can imagine that Peter documents his online nicknames in his FOAF file. Since he has more then one nickname, he adds each in a separate triple to his FOAF file:

```
http://peter.blogger.com/foaf.rdf#me foaf:nickname "peter1967" .
http://peter.blogger.com/foaf.rdf#me foaf:nickname "peterTheGreat" .
http://peter.blogger.com/foaf.rdf#me foaf:nickname "peterLikesFlowers" .
```

At some point in time, since he does not want Paula to find out about his age and his love for flowers, he changes some of his nicknames:

```
http://peter.blogger.com/foaf.rdf#me foaf:nickname "peterLikesMasculineThings" .
http://peter.blogger.com/foaf.rdf#me foaf:nickname "peter1977" .
http://peter.blogger.com/foaf.rdf#me foaf:nickname "peterTheGreat" .
```

Paula could try to use a revision control system to find out which nicknames were changed, but since RDF graphs do not have any mandated serialisation

order, the order of the triples could change arbitrarily leading to many false positives in a standard revision control system.

RDF defines literals as lexical representations of numbers and strings, which can not be addressed, because they have no identity. Therefore changes to occurrences of literals can not be tracked properly.

### 2.2 Blank nodes are not addressable

Peter wants to describe his friends in his FOAF file. He met somebody online, and only knows the person by the online nickname "tinysushi":

```
http://peter.blogger.com/foaf.rdf#me foaf:knows _:1 .
_:1 foaf:nickname "tinysushi" .
```

Paula also knows a person only by the online nickname "bigsteak":

```
http://paula.typepad.com/foaf.rdf#me foaf:knows _:76 .
_:76 foaf:nickname "bigsteak" .
```

In order to compare the acquaintances of Peter and Paula, the information from their FOAF files has to be merged. It is not possible to decide whether Peter and Paula know the same person by different nicknames or whether these are two different people.

The reason for this, is that the RDF semantics treat blank nodes only as existentially quantified variables (scoped to the entire graph) in the RDF graph in which they occur. Thus blank nodes are only unified if the two graphs contain the exact same set of assertions about them, which is in reality rarely the case. Furthermore, blank nodes cannot be referenced directly from outside their graph.

Although OWL [15] offers ways to address blank nodes by defining inverse functional properties, such indicators are domain-specific and not always available in practice. Systems giving an object-oriented representation of an RDF graph, such as ActiveRDF [16] or RDFReactor[6] have to manage blank node identity by themselves. Similarly, in SPARQL query results, blank nodes cannot be referenced in future queries.

Tracking changes in RDF graphs is relevant in many scenarios since data often changes. Ontology evolution [7] and ontology versioning systems therefore need ways to address literals, blank nodes, and statements. E. g. the ontology versioning system SemVersion [19] attaches automatically generated identifiers as inverse functional properties to each blank node, in order to identify them.

### 2.3 Statements can not be addressed

Paula finally discovers Peter's true age, through another source. Not able to change Peter's FOAF file, she wants to announce herself as the author of this discovery in her own FOAF file, using reification.

[6] http://rdfreactor.ontoware.org

```
http://peter.blogger.com/foaf.rdf#me foaf:dateOfBirth "1967/5/21"^^xsd:date .
_:9 rdf:type rdf:Statement .
_:9 rdf:subject http://peter.blogger.com/foaf.rdf#me .
_:9 rdf:predicate foaf:dateOfBirth .
_:9 rdf:object "1967/5/21"^^xsd:date .
_:9 dc:author http://paula.typepad.com/foaf.rdf#me .
```

Peter finds several FOAF files online, including Paula's, and merges them with his own. He now discovers that his real birthday is stated in one of the FOAF files. Curious about the origin of this information, he tries to find out more about it.

Given RDF semantics, there is no way for him to look up information about a given statement $(s, p, o)$ as there is formally no relation between a triple and one of its reified counterparts. Although the semantics describes informally, that applications could treat the reification triple and the real triple as the same entity, this interpretation is not mandatory and not formally specified.

Reification (mentioning a statement without asserting it) is used heavily in agent communication systems [13] that model parts of human speech, such as propositional attitude, modalities, beliefs and indirect speech. One such system is DBin [17], which allows sharing of semantically annotated information on a peer to peer basis, and supports distinguishing between information from different authors.

### 2.4 Language tags are not addressable

The RDF specification recognises the need to localise strings into different languages, and introduces the notion of "language tags" [12, Sect. 3.4]. These tags describe that a certain string literal is written in a certain language, and can be used to localise user interfaces to display information in the user's preferred language.

For example, the DOAP[7] project description schema provides labels for properties such as "programming language" and "operating system" in English, French, and Spanish. Language tags are (in N3 syntax) appended to the literals:

```
doap:os rdfs:label "operating system"@en .
doap:os rdfs:label "sistema operativo"@es .
```

These language tags are simple strings, that conform to RFC-3066[8]. As such, they are not addressable inside the RDF model: they are attached to the literal but nothing can be said about them. This leads to four problems: (a) it is impossible to indicate relations between language tags and other concepts, e. g. "nl" is used in the Netherlands, (b) it is impossible to indicate relations between language tags, such as the fact that "en" is a superset of "en-US", "en-GB" and "en-AU", (c) it is impossible for people to indicate in RDF their language preferences, except through simple strings having no formal relation to language tags, and (d) it is impossible to find e. g. all French literals, relevant when translating.

[7] `http://usefulinc.com/doap`
[8] `http://www.isi.edu/in-notes/rfc3066.txt`

### 2.5 Language tags can not be combined

Language tags are not part of the data model, they are only attachments to literals. Therefore they cannot be combined, but only used once for each string literal. For example, if we want to express that the name of a company is "Microsoft" in both English, Dutch, and Spanish, we have to use three literals:

```
http://microsoft.com# rdfs:label "Microsoft"@en .
http://microsoft.com# rdfs:label "Microsoft"@nl .
http://microsoft.com# rdfs:label "Microsoft"@es .
```

We could use a single literal if the company's name would be identical in every language, but often different company or product names are used in different regions. Thus, correctly representing Microsoft's name would lead to a high redundancy, which would lead to maintenance problems when renaming or restructuring a company or product.

## 3 Solution: SiRDF

After having introduced the problems, which are rooted in the abstract syntax and formal semantics of RDF, we will show how these problems might be addressed by the introduction of a new and simplified data model, called Simple RDF (SiRDF).

In this section, we provide an abstract syntax, formal semantics and new vocabulary for SiRDF, with the benefits of the simpler data model, while retaining the same expressivity as RDF. We explain the general principles behind SiRDF and show how the new data model allows us to solve the identified problems. Finally we proof that one can express everything in SiRDF that can be expressed in RDF.

The main idea behind SiRDF is a simplification of the RDF data model. An SiRDF graph contains only nodes and statements. To keep the expressivity of RDF language tags, datatypes and blank nodes have to be expressed through ordinary statements in the SiRDF graph. This shift from a complex data model to a simplified data model with an additional vocabulary, will lead us to suggestions for RDF best practices in 4.

**Principles** The interpretation of an SiRDF model is based, as in RDF, on a mapping from the vocabulary to the universe of the interpretation, and an extension over the universe which defines the pairs of resources for which a predicate holds true. As in RDF, this separation enables us to use the vocabulary to describe a possible state of the world, and to check, using the interpretation, whether this state is a valid state of the model. We do this by computing the truth value of all statements in the graph.

The difference to RDF is that in SiRDF every part of the graph is first mapped to a URI, which is in turn mapped to the universe. In contrast, in RDF certain entities (such as literals) are direct subsets of the universe. As a second contrast to RDF, SiRDF unifies URI references, blank nodes and literals into

one single entity: the node. Each node has an URI reference and an optional value, and every statement is identifiable by a URI reference.

**Vocabulary** SiRDF has only one kind of node and only one kind of statement. Language tags and data types are expressed in regular statements with the predicates `sirdf:datatype` and `sirdf:languagetag`. For reification, a node is declared as `rdf:type sirdf:Statement`. Then, similarly to the RDF reification vocabulary, this node can use the predicates `sirdf:subject`, `sirdf:predicate`, and `sirdf:object`. To express our examples, we will use the following notation for each SiRDF statement in this paper:

```
statementURI (subjectURI "optional value" predicateURI "optional value" objectURI "optional value").
```

### 3.1 Solving the Problems of RDF

We will now show, how SiRDF addresses the earlier identified issues. We illustrate our points by applying SiRDF to the introduced example scenario: Peter and Paula migrate their FOAF files to SiRDF.

**Occurrence of literals is addressable** Peter puts all of his nicknames into his FOAF file using SiRDF, which results in the following statements:

```
peter:statement1 ( peter:me foaf:nickname peter:literal1 "peter1967" ).
peter:statement2 ( peter:me foaf:nickname peter:literal2 "peterTheGreat" ).
peter:statement3 ( peter:me foaf:nickname peter:literal3 "peterLikesFlowers" ).
```

After publishing the new file for the first time, he remembers Paula and changes two literals:

```
peter:statement3 ( peter:me foaf:nickname peter:literal3 "peterLikesMasculineThings" ).
peter:statement1 ( peter:me foaf:nickname peter:literal1 "peter1977" ).
peter:statement2 ( peter:me foaf:nickname peter:literal2 "peterTheGreat" ).
```

With SiRDF Paula can keep track of changes to a certain literal, because every occurrence of a literal is identified by a URI reference. Paula now can see that "peter1967" changed to "peter1977" and that the flowers are replaced by masculine things.

SiRDF does not distinguish between URI references and literals: both concepts have been unified into the new concept of a node. Each node has the properties of both, a URI reference and a value to hold the lexical representation of a literal. The value can be left empty, so that only the URI reference of the node is used. Therefore every value of a node can be identified by the URI reference of the same node.

**Blank nodes are addressable** Blank nodes have been merged into the concept of the node, together with URI references and literals. Thus, when Peter first meets "tinysushi" online, he wants to record this in his SiRDF FOAF, without using a blank node. Instead of creating a URI reference as a place holder he asks "tinysushi" for a URI, and uses that:

```
peter:statement4 ( peter:me foaf:knows hank:me ).
peter:statement5 ( hank:me foaf:nickname peter:literal4 "tinysushi" ).
```

Paula meets somebody called "bigsteak" online, but she gets no other details about the person. She asks the person about his FOAF file, and puts the information into her own FOAF file.

```
paula:statement1 ( paula:me foaf:knows hank:me ).
paula:statement2 ( hank:me foaf:nickname paula:literal1 "bigsteak" ).
```

If the information from the FOAF files of Peter and Paula get merged, they will discover that they know the same person and that this person is known by two different nicknames. An SiRDF node can be used instead of a blank node. The URI reference of the node can be used to identify the node, including from another graph. Eliminating blank nodes by mandating the exclusive usage of URI references, is the same as renaming all blank nodes with URI references through skolemisation, as suggested by the RDF semantics.

**Statements can be addressed** Paula can publish Peter's true age in her SiRDF FOAF file. Since Paula can address every statement made by her or others, she can express her authorship easily. When Peter merges his FOAF file with Paula's file, he can discover the author of that statement directly, by querying for statements with `paula:statement3` as the subject.

```
paula:statement3 ( peter:me foaf:dateOfBirth paula:literal2 "1967−05−21" ).
paula:statement4 ( paula:literal2 "1967−05−21" sirdf:datatype xsd:date ) .
paula:statement5 ( paula:statement3 dc:author paula:me ).
```

The reason for this, is that every SiRDF statement must have an URI reference, which gives it an identity and which makes it addressable. Note that the same statement may appear multiple times with different URIs. Formally, the statement is simply true. On a higher level, this enables different sets of annotations about the statements.

**Language tags are addressable** Instead of introducing new model elements, SiRDF expresses language tags and datatypes of literals through a special vocabulary. Since all literals in SiRDF are addressable we can make statements about them, such as their datatype or language. The language tags themselves are URIs, similarly to earlier designs[9][10].

Paula can express in SiRDF, that American and British English are forms of English, she can say that Dutch is spoken in the Netherlands (using the geonames ontology), and indicate her own proficiency in Dutch:

```
paula:statement6 ( lang:en−US rdfs:subClassOf lang:en ).
paula:statement7 ( lang:en−GB rdfs:subClassOf lang:en ).
paula:statement8 ( lang:nl lang:spokenIn http://sws.geonames.org/2750405/ ).
paula:statement9 ( paula:me lang:reads lang:nl ).
```

[9] `http://www.w3.org/DesignIssues/InterpretationProperties.html`
[10] `http://www.ninebynine.org/IETF/URNs/draft-klyne-urn-ietf-lang.html`

**Language tags can be combined** Again, since on the one hand, language tags are not data model elements but are expressed similar to any other property, and on the other hand, all literals have a URI, we are now able to express that the same literal applies to several languages:

```
paula:statement10 ( http://microsoft.com# rdfs:label paula:mslabel "Microsoft" ).
paula:statement11 ( paula:mslabel lang:lang lang:en ).
paula:statement12 ( paula:mslabel lang:lang lang:es ).
paula:statement13 ( paula:mslabel lang:lang lang:nl ).
```

### 3.2 Proof of Equivalence

We will now provide the formal definitions of SiRDF. Afterwards we will show that it is possible for an SiRDF model and an RDF model to have the same denotation and the same interpretation.

**Equivalence sketch** We first sketch the main part of that equivalence proof, by describing how each RDF model element relates to an SiRDF model element. The full conversion algorithm, in both directions, is given in [11]. As shown in table 1, the conversion from RDF to SiRDF is straightforward: each URI reference, blank node, and literal is represented with an SiRDF node. For blank nodes and literals, new URIs have to be given or automatically created. Nodes stemming from literals have a value, other nodes have no value. Language tags and datatypes are attached to (the occurrence of) a literal using regular statements.

| RDF | SiRDF | explanation |
|---|---|---|
| `<uri>` | `(uri)` | node without value |
| `_:#1` | `(bnode-uri)` | unvalued node, with new URI |
| `"literal"` | `(literal-uri "literal")` | node with new URI and value |
| `"literal"@lt` | `literal-uri lang:lang lang:lt .` | statement about literal's language tag |
| `"literal"4ype` | `literal-uri sirdf:datatype type .` | statement about literal's datatype |

**Table 1.** Converting between RDF and SiRDF

**SiRDF formal definition** We now briefly introduce the data model, abstract syntax, and formal semantics of SiRDF.

**Definition 1 (SiRDF graph).** *An SiRDF graph consists of nodes and statements. A statement has a unique identifier and contains three nodes: a subject, predicate and object. Each node has a unique identifier and optionally a value.*

**Definition 2 (Abstract syntax).** *An SiRDF graph $G = (S, N)$ contains the set of statements $S$ and the set of nodes $V$. A node $N = (U, V)$ consists of a URI $U$ and a value $V$, where $U$ and $V$ are disjoint subsets of the vocabulary on*

*which the graph is defined. A statement is* $S_t = (U, \{N, S_t\} \times \{N, S_t\} \times \{N, S_t\}$, *where* $U$ *is a URI reference, each triple element (subject, predicate, and object) is either a node or a statement.*

We now define the formal semantics which are as close to their RDF counterparts as possible, while taking the simplifications of the SiRDF data model into account. We start with the definition of the constituents of the interpretation $I()$. The mappings constitute the interpretation and describe a model of reality.

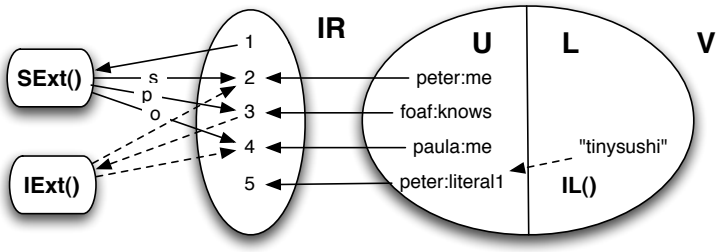


**Figure 1.** Illustration of the Interpretation of an SiRDF Graph

**Definition 3 (Interpretation of an SiRDF graph).**
*The vocabulary* $V$ *is the set* $V = U \cup L$, $U$ *and* $L$ *are disjoint.* $U$ *is the set of URI references. Every node or statement is identified by an URI reference* $u \in U$. $L$ *is the set of literals (or values).* $IL : L \longmapsto U$ *is an injective function, which assigns every literal to an URI reference.* $SExt : U \longmapsto (U \times U \times U)$ *is an injective function from the set* $U$ *of URI references into the set of triples, constructed from* $U$.

*The interpretation* $I$ *on the vocabulary* $V$ *is defined by: the set* $IR$ *(universe) of all resources in* $I$*; the injective function* $IS : U \longmapsto IR$ *from the URI references onto the universe of* $I$*; and the mapping* $IExt : IR \longmapsto (IR \times IR)$ *from the universe* $IR$ *onto* $IR \times IR$*, the set of all pairs in* $IR$.

We now define the denotation of an SiRDF graph, which is a truth value based on the interpretation of all statements in the graph.

**Definition 4 (Denotation of a SiRDF node).** *Let* $E$ *be a node with value* $W$ *and the URI reference* $U$*, then* $I(E) = IS(U)$ *(applied to the parts* $W$ *and* $U$ *of the node,* $I(E) = I(U) = I(W) = IS(U)$ *holds true).*

**Definition 5 (Denotation of a SiRDF statement).** *Let* $E$ *be a statement with the URI reference* $U$ *and the URI references* $s$, $p$, $o$ *which get resolved through* $SExt(U) = (s, p, o)$*. Then the denotation of* $E$ *is given through:*

$$I(E) = \quad IS(s), IS(p), IS(o) \in IR \tag{1}$$

$$\wedge (IS(s), IS(o)) \in IExt(IS(p)) \tag{2}$$

**Definition 6 (Denotation of a SiRDF graph).** *Let $E$ be an SiRDF graph, $I(E) = false \iff \exists a \in E : I(a) = false$, otherwise $I(E) = true$.*

**Proof of expressive equivalence** We will now show in a formal way that it is possible to express everything in SiRDF, that can be expressed in RDF. The formal semantics of SiRDF have been constructed to be as similar to RDF as possible. This will be used to define what conditions have to be meet in order for an SiRDF and an RDF graph to have the same interpretation and denotation.

By basing the interpretation of an SiRDF graph on the interpretation of an RDF graph, we can define the conditions under which the denotation and interpretation of an SiRDF graph and an RDF graph are equal:

**Definition 7 (Conditions for equal interpretation of an SiRDF graph and of an RDF graph).**

*Let $I_R$ be an interpretation over an RDF graph, defined as in [10, Sect. 1.3]: $I_R$ is based upon the vocabulary $V$, the universe $IR$ and the properties $IP \subset IR$, which are defined through the mappings $IExt()$, $IS()$ and $IL()$. The extension of $I$ to handle blank nodes is $A()$. The denotation of the RDF graph then is given through the definitions in [10, Sect. 1.4] and [10, Sect. 1.5].*

*Let further $I_S$ be an interpretation of an SiRDF graph, as in definition 3. Parts of $I_S$, that are equal in $I_R$: the vocabulary $V$, the universe $IR$, the mapping $IExt()$ from predicates to pairs in the universe $IR$ (which describes whether a predicate holds true over a subject and object) and the mapping $IS()$ from URI references to $IR$, which maps URI references to "things" in the universe.*

*Parts of $I_S$, that are different from $I_R$: the vocabulary $V$ is split into the disjoint subsets: $U$ of URI references in $V$ and $L$ of values in $V$; the mapping $IL_S()$, which maps every value from $L$ to its corresponding URI reference in $U$; the mapping $SExt()$, which maps the URI reference addressing a statement, to the URI references of its subject, predicate and object.*

*Given these conditions, the interpretations $I_R$ and $I_S$ have the same truth values.*

Now we will prove that, given these conditions, the interpretation of an SiRDF graph and of an RDF graph are the same. This means, that it is possible to express everything in SiRDF, that can be expressed in RDF, and vice versa.

**Definition 8 (SiRDF $\implies$ RDF).** *Let $I_S(a) = true$ hold for a statement $a$ from an SiRDF graph, and let $s$,$p$,$o$ be defined through $SExt(a) = (s, p, o)$. Then $I_R(spo) = true$ iff:*

1. *$s, p, o \in V$: $SExt()$ maps every statement $a$ to a triple of uri references, and because $U \subset V$ the condition holds.*
2. *$I_R(p) \in IP$: Because $p \in U$, then also $I_R(p) = I_S(p) = IS(p) \in IP \subset IR$*
3. *$(I_R(s), I_R(o)) \in IExt(I_R(p))$: Because $s, o \in U$, then also $(I_R(s), I_R(o)) = (IS(s), IS(o)) = (I_S(s), I_S(o))$. Further $I_R(p) = I_S(p) = IS(p)$ and also $(I_R(s), I_R(o)) \in I_S(p)$, therefore: $(I_R(s), I_R(o)) \in I_R(p)$*

**Definition 9 (RDF $\Longrightarrow$ SiRDF).** *Let $I_R(s,p,o) = true$ for a triple $(s,p,o)$ in the RDF graph. If the SiRDF graph contains a statement $a$ with $SExt(a) = (s,p,o)$, then $I_S(a) = true$ iff: $IS(s), IS(p), IS(o) \in IR \wedge (IS(s), IS(O)) \in IExt(IS(p))$. Since every predicate is a URI reference, $IS(P) \in IR$. Let $n$ be the subject $s$ or the object $o$, then depending on the type of $n$:*

- *URI reference: $IR$ is the same for $I_S$ and $I_R$, so $I_R(n) = I_S(n)$*
- *simple literal: Let $E$ be a simple literal. Choose $u \in U$ so that $IL_S(p) = u$ and $IS(u) = p$. Then $IS(n) = IS(U) = IS(IL_S(n)) \in IR$ holds true.*
- *literal with datatype or language tag: Let $E$ be a literal with a datatype or language tag. This is a given: $IL_R(n) \in IR$. Now we can choose $u \in U$ so that $IL_R(n) = IS(u)$ and $IL_S(n) = u$. Then $IL_R(n) = IS(u) = IS(IL_S(n)) \in IR$ holds true.*
- *blank node: Let $E$ be a blank node. Then for $I_R$ as given in [10, Sect. 1.5]: $[I_R + A](n) = A(n)$ and $A(n) \in IR$. Choose $u \in U$ so that $IS(u) = A(n)$. Then $I_S(n) = IS(u) = A(n) \in IR$ holds true.*

### 3.3 Merging SiRDF graphs

An important feature of RDF graphs is the possibility of merging: graphs without blank nodes are merged through the set union of their triples, while for graphs with blank nodes the set union is performed while ignoring the blank node identifiers. To merge two SiRDF graphs $S_1$ and $S_2$, we first translate both to their RDF counterparts $R_1$ and $R_2$, merge those to $R_3$, and translate that graph back to the SiRDF graph $S_3$.

## 4 Solution: Restricted RDF

Since changing the RDF standard is not a practical solution, this section contains the most practical contribution of this paper: we show how to represent the SiRDF-ideas using the given RDF abstract syntax and show how these recommendations resolve the introduced problems. SiRDF has demonstrated that a simpler data model can have the same expressivity as a richer data model, if the missing data model parts can be expressed through usage of the vocabulary. The main idea behind our recommendations, is to restrict the usage of the RDF data model.

### 4.1 Addressing the occurrence of literals

Peter is eager to try out restricted RDF, and he puts all his nicknames in a FOAF file using RDF, but seeing SiRDF has given him an idea:

```
http://peter.blogger.com/foaf.rdf#me foaf:nickname http://peter.blogger.com/foaf.rdf#literal1 .
http://peter.blogger.com/foaf.rdf#literal1 rdf:value "peter1967" .
http://peter.blogger.com/foaf.rdf#me foaf:nickname http://peter.blogger.com/foaf.rdf#literal2 .
http://peter.blogger.com/foaf.rdf#literal2 rdf:value "peterTheGreat" .
http://peter.blogger.com/foaf.rdf#me foaf:nickname http://peter.blogger.com/foaf.rdf#literal3 .
http://peter.blogger.com/foaf.rdf#literal3 rdf:value "peterLikesFlowers" .
```

Changes to his nicknames can now be tracked by the URI references to which these literals are attached. Peter can use the `rdf:value` term, although it currently has no formal meaning in the RDF semantics. Using `rdf:value` we can thus address literals. In fact, this solution is very close to the solution in SiRDF, which combines literal values and URI references in the concept of a node. A URI with an `rdf:value` can actually be considered both a URI and a literal, and interpreted similarly to the interpretation in SiRDF.

### 4.2 Addressing blank nodes

Peter wants to add somebody to his FOAF file, about whom he knows only the nickname "tinysushi". Restricted RDF discourages usage of blank nodes, so Peter has to ask his friend for his URI:

```
http://peter.blogger.com/foaf.rdf#me foaf:knows http://hank.vox.com/foaf.rdf#me .
http://hank.vox.com/foaf.rdf#me foaf:nickname "tinysushi" .
```

Paula meets somebody called "bigsteak" online, and also asks him for a URI:

```
http://paula.typepad.com/foaf.rdf#me foaf:knows foaf:knows http://hank.vox.com/foaf.rdf#me .
http://hank.vox.com/foaf.rdf#me foaf:nickname "bigsteak" .
```

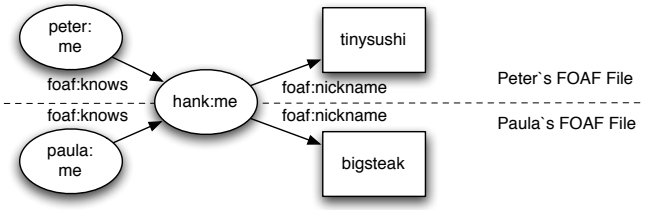


**Figure 2.** Discouraging blank nodes can result in better information sharing

When merging their files, Peter and Paula will again discover that they know the same person under two different nicknames. Peter and Paula could also have chosen two different URIs for their friend, and would not so easily find out that he is the same person. If they do find this out later, they can represent that with vocabulary such as `owl:sameAs` and `owl:InverseFunctionalProperty` [15].

### 4.3 Reification semantics: enabling statement addressing

Paula wants to state Peter's date of birth in her FOAF file, but in a responsible way. So she will declare herself as the author of this information:

```
http://peter.blogger.com/foaf.rdf#me foaf:dateOfBirth "1967/5/21"^^xsd:date .
http://paula.typepad.com/foaf.rdf#42 rdf:type rdf:Statement .
http://paula.typepad.com/foaf.rdf#42 rdf:subject http://peter.blogger.com/foaf.rdf#me .
http://paula.typepad.com/foaf.rdf#42 rdf:predicate foaf:dateOfBirth .
http://paula.typepad.com/foaf.rdf#42 rdf:object "1967/5/21"^^xsd:date .
http://paula.typepad.com/foaf.rdf#42 dc:author http://paula.typepad.com/foaf.rdf#me .
```

After merging the information from multiple FOAF files, and discovering his birth date, Peter will want to know, who is the author of this information. If a reification triple and a real triple with the same content exist, these should be linked by the formal semantics. Then Peter can look for anybody claiming authorship of the statement and decide whether to trust this information.

The missing part in the (formal) RDF semantics specification, is the association between the triple, the reified triple of the same contents and the URI reference of the reified triple. Formally stated, these three entities in the graph have to denote the same thing in the universe. Another approach to give addressing semantics to statements, would be to use named graphs [3], which are commonly used to track provenance. If each named graph would contain only one triple, the name of the graph could be used as address of the triple itself, allowing efficient reification. Both named graphs and reification can be implemented using quad patterns (as in YARS [9]).

### 4.4 Language tags are addressable

Regarding the language tags, we can directly apply the solution of SiRDF to common RDF. Instead of using language tags as distinct model elements we introduce a simple vocabulary for these tags, promote language tags to normal URIs, and use normal triples to express language-related information:

```
lang:en−US rdfs:subClassOf lang:en .
lang:en−GB rdfs:subClassOf lang:en .
lang:nl lang:spokenIn http://sws.geonames.org/2750405/ .
paula:me lang:reads lang:nl .
```

### 4.5 Language tags can be combined

Since we can address literals, through the explained usage of `rdf:value`, we can describe various properties of the same literal, allowing us to combine language tags for a single literal without repeating the literal:

```
http://microsoft.com# rdfs:label paula:mslabel .
paula:mslabel rdf:value "Microsoft" .
paula:mslabel lang:lang lang:en .
paula:mslabel lang:lang lang:es .
paula:mslabel lang:lang lang:nl .
```

## 5 Related work

Other extensions to RDF have been proposed. Mazzieri [14] describes how to augment RDF with fuzzy semantics. Gutierrez [8] restricts the truth value of a statement for given time intervals. Similar to our approach, they introduce vocabularies to represent the additional semantics as RDF statements; on the other hand, they target the specific problem domains of fuzzy and temporal reasoning. The idea to address literals can be found in the RDF specification

itself, as each literal entails the existence of a blank node [10, Sect. 4.3]. Early implementations of RDF used this technique internally[11]. We follow this idea and mandate such usage.

Tummarello *et al.* [18] and Ding *et al.* [6] both suggest methods for splitting a graph into self-describing fragments, which then can be compared to decide if merging is possible. These two approaches solve the problem of merging incomplete information for some cases, but not all; we argue that this problem should be solved before putting the information into an RDF graph.

We identified the need to mandate a link between a statement and its reification statement, but we did not provide a formal semantics for it. Conen *et al.* [5] add identification semantics to reified statements and prove the compatibility with current RDF semantics; we see this work as complementary to ours.

Finally, treating language tags as first class entities in the graph, has been advocated before by Carroll and Phillips [4]. Their solution relies partly on OWL, and therefore is more expressive but also more complex then ours.

## 6 Conclusions

We have analysed and discussed five modelling issues in RDF. We resolved these issues by simplifying the RDF data model into SiRDF. SiRDF moves language tags and data types from the model into the vocabulary, and makes all literals and statements addressable, while keeping the same expressivity as RDF.

SiRDF is simpler than RDF: there are less concepts to learn and less concepts to implement. By giving every concept an identifier we also enhance the meta-modelling capabilities of RDF: one can now for example describe relationships between language tags. Since changing the already deployed RDF infrastructure is unrealistic, we then presented restricted RDF as a way of modelling the simplified data model of SiRDF with a subset of the RDF data model; the missing parts are expressed through a new vocabulary. Table 2 summarises the recommended changes and the responsibilities for implementing the recommendations.

| *element* | *change* | *responsibility* |
|---|---|---|
| URI reference | unchanged | – |
| blank node | discouraged | modeller |
| literal | use `rdf:value` | modeller |
| language tag | use `lang:lang` | modeller |
| datatype | use `sirdf:datatype` | modeller |
| reification | mandatory semantics | implementations |

**Table 2.** Overview of changes

[11] `http://lists.w3.org/Archives/Public/www-rdf-interest/2001Feb/0090.html`

## References


[1] D. Brickley and R. Guha, (eds.) *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Candidate Recommendation, 2004.

[2] J. de Bruijn, H. Lausen, A. Polleres, and D. Fensel. The web service modeling language WSML: An overview. In *European Semantic Web Conference*, pp. 590–604. 2006.

[3] J. J. Carroll, C. Bizer, P. Hayes, and P. Stickler. Named graphs, provenance and trust. In *International World Wide Web Conference*, pp. 613–622. 2005.

[4] J. J. Carroll and A. Phillips. Multilingual RDF and OWL. In *European Semantic Web Conference*, pp. 108–122. 2005.

[5] W. Conen, R. Klapsing, and E. Köppen. RDF M&S revisited: From reification to nesting, from containers to lists, from dialect to pure XML. In *Semantic Web Working Symposium*, pp. 195–208. 2001.

[6] L. Ding, T. Finin, Y. Peng, P. da Silva, *et al.* Tracking RDF graph provenance using RDF molecules. In *International Semantic Web Conference (poster paper)*. 2005.

[7] A. Gomez-Perez, O. Corcho, and M. Fernandez-Lopez. *Ontological Engineering*. Springer, 2005.

[8] C. Gutierrez, C. Hurtado, and A. Vaisman. Temporal RDF. In *European Semantic Web Conference*, pp. 93–107. 2005.

[9] A. Harth and S. Decker. Optimized index structures for querying RDF from the web. In *Proceedings of the 3rd Latin American Web Congress*. 2005.

[10] P. Hayes, (ed.) *RDF Semantics*. W3C Recommendation, 2004.

[11] B. Heitmann. *SiRDF - Ein simpleres Datenmodell für RDF bei gleicher Ausdrucksmächtigkeit, Konzeptionelles Modell und Java API*. Studienarbeit, Institut AIFB, Universität Karlsruhe, 2005.

[12] G. Klyne and J. J. Carroll, (eds.) *Resource Description Framework: Concepts and Abstract Syntax*. W3C Recommendation, 2004.

[13] Y. Labrou, T. Finin, and Y. Peng. Agent communication languages: the current landscape. *IEEE Intelligent Systems*, 14(2):45–52, 1999.

[14] M. Mazzieri. A fuzzy RDF semantics to represent trust metadata. In *Semantic Web Applications and Perspectives (online proceedings)*. 2004.

[15] D. L. McGuinness and F. van Harmelen, (eds.) *OWL Web Ontology Language*. W3C Recommendation, 2004.

[16] E. Oren, R. Delbru, S. Gerke, and A. Haller. ActiveRDF: Object-oriented semantic web programming. Tech. Rep. 2006-13, DERI Galway, 2006.

[17] G. Tummarello, C. Morbidoni, and M. Nucci. Enabling semantic web communities with DBin: An overview. In *International Semantic Web Conference*, pp. 943–950. 2006.

[18] G. Tummarello, C. Morbidoni, P. Puliti, and F. Piazza. Signing individual fragments of an RDF graph. In *International World Wide Web Conference*, pp. 1020–1021. 2005.

[19] M. Völkel and T. Groza. Semversion: An RDF-based ontology versioning system. In *IADIS International Conference on WWW/Internet*, pp. 195–202. 2006.