Achieving Scalability and Expressivity in an RDF Knowledge Base by Implementing Contexts

Heiko Stoermer
University of Trento
Dept. of Information and Communication Tech.
Trento, Italy
Email: stoermer@dit.unitn.it

Ignazio Palmisano
Università degli Studi di Bari
Dipartimento di Informatica
Bari, Italy
Email: palmisano@di.uniba.it

Domenico Redavid
Università degli Studi di Bari
Dipartimento di Informatica
Bari, Italy
Email: redavid@di.uniba.it

***Abstract*— In this paper we are presenting the context architecture implemented on top of the *RDFCore* system. With this extended Knowledge Representation framework we are trying to overcome some of the limitations of RDF and OWL as they are today, without losing sight of performance and scalability issues. We are illustrating motivations – partly based on requirements in the VIKEF project – as well as theoretical background, implementation details and test-results of our latest works.**

I. Introduction

Motivated by requirements of the VIKEF[1] project, where a large-scale Semantic Web knowledge-base about documents and other objects provides for intelligent services to the user, we are investigating and developing a more extended KR framework, trying to overcome some of the limitations of RDF and OWL as they are today.

Our basic idea is to introduce the notion of context into Semantic Web Knowledge Representation (KR), as previously described in [2], [14]. We claim that the distributed nature of the Semantic Web raises issues that can be attacked by contextualizing knowledge bases, i.e. restricting the scope of statements to the circumstances they were made under.

The contribution of this paper is to present one possible realization of this more complex KR approach for the Semantic Web, and to illustrate our progress based on the KBMS *RDFCore* [4]. In continuation of the ideas and preliminary results presented in [14], we have been concentrating more on the aspects of compatibility relations (CRs) between contexts, which can be used to describe in which way statements in more than one context can be combined to answer queries to the KB. We have conducted a more extensive experiment to investigate performance aspects of *RDFCore* and our extensions, and backed by general theories of Contextual Reasoning we believe that we will in some cases be able to provide for better scalability than a flat, non-contextual KB.

The paper is organized as follows: In Sect. II we present intuitive and technical motivations for our approach, as well as related work. Sect. III describes our general proposal, whereas Sect. IV contains a technical description of the steps taken to realize our ideas. In Sect. V we present our experimentation results, and we wrap up with a conclusion and a short mention of planned further works in Sect. VI.

[1]Virtual Information and Knowledge Environment Framework; more information at `http://www.vikef.net`

II. Motivation and Related Work

One of our initial motivations to move in the direction of contexts in Semantic Web KBs was our critical view on one of the ideas of the Semantic Web, namely that – with a shared ontology – two RDF Aboxes provided by different agents can simply be merged, collapsed on identical URIs, and thus provide a new, bigger KB for answering a query (the pre-merge scenario is depicted in Fig. 1).

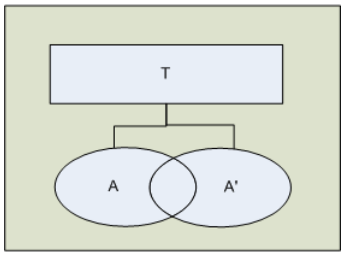


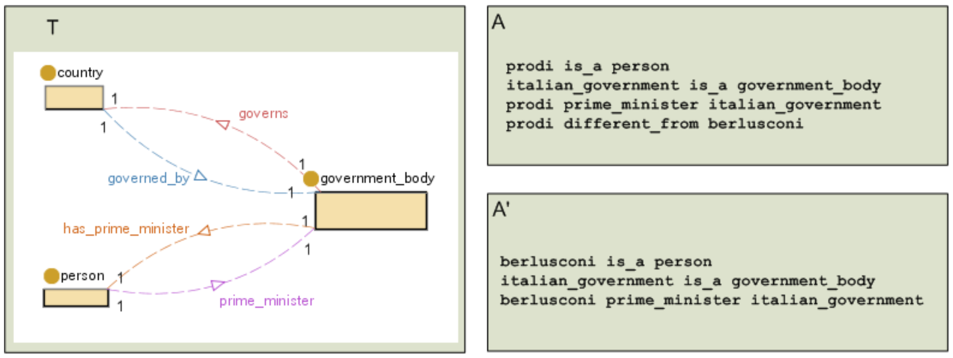
Fig. 1. Two RDF Aboxes A and A' compliant to a single TBox T.

However, apart from implicit semantics that are omitted when applying such a strategy, cases can be constructed that unveil problems even on the logical level. Take the following example, as depicted in Fig. 2: on the formalization side we have a TBox T with some relations that have cardinality constraints, and two ABoxes A and A' with assertions compliant to this TBox. Both ABoxes are consistent by themselves, but when merged, they produce an inconsistency as the two following statements violate the cardinality constraints in T:

$< prodi\ prime\_minister\ italian\_government >$
$< berlusconi\ prime\_minister\ italian\_government >$

Relying on a host of research done in the area of Context in KR [7], [12], [11], [6], [1], [5], [13], we believe it is a viable approach to attack issues of this nature by binding consistent sets of assertions to the circumstances they were made under, i.e. to limit their scope to a context, as we will describe in Sect. III.

As discussed in [8], [2], [3], this contextualization can serve as a basis for a number of KR modelling aspects, such as temporal evolution, trust, beliefs and provenance. The



```
prodi is_a person
italian_government is_a government_body
prodi prime_minister italian_government
prodi different_from berlusconi
```

```
berlusconi is_a person
italian_government is_a government_body
berlusconi prime_minister italian_government
```

Fig. 2. Example formalization that produces an inconsistency when merged.

contributions of our approach compared to the proposals made in [8], [3], [10], [9] as well as compared to named graph implementations in current RDF triple stores are that i) we do not propose or require an extension of the current RDF standard and ii) we aim at substantial support for Compatibility Relations (CRs).

These relations between contexts enable us to make explicit in which way the assertions in the related contexts are supposed to be combined for query answering, to provide for flexible and powerful contextual reasoning as envisioned in the mentioned bibliography.

In the course of the VIKEF project it became evident that some of the relations we have in mind have procedural semantics, and can thus not be formalized in an OWL ontology, and these are what we are concentrating on at the moment. In the next section we will describe our examplary proposal of such a complex relation.

III. An Exemplary CR

The EXTENDS relation we have chosen to illustrate is meant to describe a situation where we know that two contexts describe the same object, but assume that one context contains more information about it than the other.

Take the example of two Information Extraction processes P and P' that are run on the same document, at different points in time. Assume P' is a more advanced process and is able to extract more information from the document. We propose to model this as two contexts C (created by P) and C' (created by P') with a relation $EXTENDS$ that explicates that C' is an extension to C (a necessary condition for this relation is that both contexts describe the same object). Intuitively we want to keep the information derived from different sources separate and with explicit metadata, but have the possibility to combine the resulting information where necessary.

When a query q is posed on C', the procedural semantics of $EXTENDS$ are envisioned as follows:

```
if q can be answered in C′
    then return answer
else
    propagate query to  C′ union C.
```

One issue that becomes obvious immediately is the case where the union of C' and C produces an inconsistent ABox which makes query-answering impossible. This can result from cardinality constraints in the TBox (see the Berlusconi-Prodi example in Sect. II), or subsumption issues (an individual o is said to be instance of different classes). Our basic solution approach is to extract a minimal subgraph containing the statement(s) that caused the inconsistency into a named graph NG, as illustrated in Fig. 3.

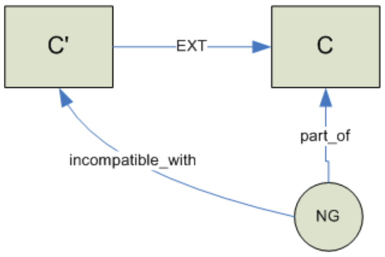


Fig. 3. Two contexts C and C' in an EXTENDS relation.

The result is that the query can be processed on the conflict-free part of the union of C' and C. One possible criticism could be that of course we could pose the query to C alone, without respecting C', and thus avoid the conflict altogether. This however ignores the $EXTENDS$ relation between the two (which has been established for a reason), and thus should only be allowed on contexts that are not in such a relation.

The case is of course slightly more complex when we take into account more than two contexts. We envision the $EXTENDS$ relation to be transitive. This can result in a reasoning chain i) when establishing the relation, as conflicts have to be detected and re-modelled and ii) when querying the contexts, as the necessary contexts and relevant subgraphs have to be traversed. This chain however is non-cyclic, as the relation is directional. Section IV describes our first implementation of this relation.

We have chosen to attack and illustrate this specific relation due to its relative complexity. However, we are convinced that our basic approach as described in [14] is fairly general and can be used to implement relations of different kinds. In the course of the project we envision relations that make explicit temporal evolution, trust and a number of domain specific aspects.

IV. Realization

A. *RDFContextManager*

The component we developed to manage contexts is called *RDFContextManager*; its architecture is presented in Fig. 4. *RDFContextManager* is implemented as a Java interface, exposing methods to:

- set the Compatibility Relation Ontology (CRO), which is the ontology that defines Compatibility Relations, Contexts, parts of Contexts (Graphs) and also gives the concepts necessary to represent context splitting and relations between Contexts and Graphs
- add new statements to the CRO, stating for example that a given URI C_1 represents a Context, that this Context extends an existing context C_2, or that there is a Graph G_1 which is part of C_2 and is compatible or not with C_1
- add, remove or update Contexts and Graphs in the underlying persistence layer
- obtain Views over a Context, e.g. ask *RDFContextManager* to return all Contexts and Graphs that are connected to a Context C_1 with EXTENDS relations, directly or by means of *part_of* relations, following all the relation chains and obeying imposed limitations

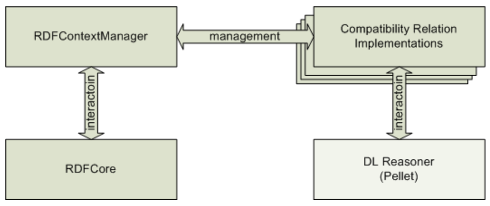


Fig. 4. Architecture of RDFContextManager

A *CompatibilityRelation* is a Java interface exposing methods to:

- verify whether an implementation of *CompatibilityRelation* should be triggered into action by some statements added to the CRO, e.g. the insertion of a statement C_1 EXTENDS C_2 should trigger the consistency check over $C_1 \sqcup C_2$, and, if an inconsistency is detected, countermeasures should be undertaken, in order to guarantee that a View over C_1 do not answer an inconsistent set of statements
- carry out the check specific for this *CompatibilityRelation*
- ask this implementation to provide a set of Contexts or Graphs that would be excluded from a View over a Context C_1 due to some reason, e.g. incompatibility due to inconsistency
- ask the implementation to provide a set of Contexts or Graphs that would be included in a View over a Context C_1, e.g. because of an EXTENDS or a *part_of* relation or chain of relations

The implementation we are presenting in this paper relies on *RDFCore* for RDF models storage, and on Pellet[2] for reasoning tasks such as consistency check over a View. As illustrated in Fig. 4, the DL reasoner is used by the *CompatibilityRelation* implementations (note that different implementations could need different reasoning settings, e.g. only RDFS or OWL Lite inference rather than OWL DL inference), while all the storage and retrieval of RDF models is done on *RDFContextManager*, which uses *RDFCore* and its facilities for model storage and query[14], using the multiuser environment of *RDFCore* to enable use of Context information by other applications.

B. *The Compatibility Relations Ontology (CRO)*

The CRO contains the definition of the main concepts used to describe the KB structure in terms of contexts; it contains the definition of Context and the definition of Graph, where both concepts represent entities that are named graphs; a Context has the (informal) property of representing something that has a meaning as a whole, e.g. the set of statements extracted from a specific document, at a specific time, with a specific algorithm, while a Graph is a set of statements that is included in one or more Contexts or other Graphs, but has no specific meaning alone (e.g. the set of statements in a Context that cause inconsistencies with another Context). A domain-range view of the CRO is given in Fig. 5.

Moreover, the CRO contains the definition of the SplittingReason class, which represents the reason that led to the isolation of a part of a Context and the storage of that fragment as a Graph; a SplittingReason instance includes references to the Context from where the statements that are being split belonged, the Graph that will hold these statements, the reason for which this split has been done, e.g. because the statements create inconsistencies w.r.t. another context (which is also linked to the reason), and the reification of the statements in the CRO that triggered the split, if any.

An example of SplittingReason generation is the one we will illustrate in detail in Sect. IV-C.1: let us have Contexts C_1 and C_2, if we add to the CRO the statement $S_1 = C_1$ EXTENDS C_2, this will trigger a consistency check over $C_1 \sqcup C_2$. If there is an inconsistency, the statements in C_2 that cause the inconsistency are moved to a Graph G_1, and then a SplittingReason SR_1 will be created in the CRO, linked to C_2 and G_1, with a reason of class Inconsistency which is linked to C_1 and a *part_of* relation between C_2 and G_1; S_1 will be reified and attached to SR_1, so that the complete splitting process can be tracked.

The CRO also acts as a registry for *CompatibilityRelation* implementations, since each declaration of a *CompatibilityRelation* amounts to the declaration of a property in this ontology; an AnnotationProperty for this property, called *implementation_uri*, gives the java class name of the corresponding implementation; this is used to retrieve the set of *CompatibilityRelation* that *RDFContextManager* will use when managing the CRO and the knowledge base.

[2]`www.mindswap.org/2003/pellet/`

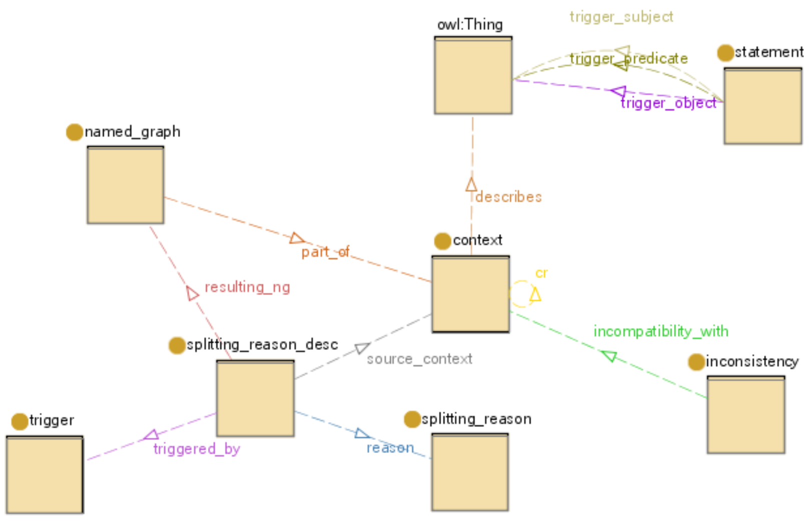


Fig. 5. Domain-range view of the CR Ontology

C. Use of Compatibility Relations (CR)

The simplest use case for the framework is as follows:

- An external application adds one or more different Contexts in *RDFContextManager*, assigning them URIs or letting *RDFContextManager* choose one
- The external application asserts some relations between the contexts or specific to a context; the relations between the contexts are expressed through properties defined in the CRO
- *RDFContextManager* receives these new assertions, and triggers all the CR implementations available into first verifying if any of the new assertions is relevant (i.e. the asserted relation corresponds to the URI the implementation is attached to) and then checking whether the new relation is likely to cause reorganization of the knowledge base; if this is the case, corrective actions are undertaken
- The external application makes a query over the CRO to find out all the contexts that satisfy some conditions (e.g. all the contexts which have been created in a specific date), and then asks to perform a query over the set of statements resulting from the union of the contexts; this involves creation of a View for each context that is selected by the query

1) EXTENDS Example: We will now use EXTENDS as a practical example of the described use case:

- Two contexts C_1 and C_2 are inserted in *RDFContextManager*
- C_1 is asserted to extend C_2 w.r.t a specific subject S_1: the following statements are added to the CRO:
  $< C_1\ EXTENDS\ C_2 >$
  $< C_1\ describes\ S_1 >$
  $< C_2\ describes\ S_1 >$
  Matching objects for the $describes$ predicate are necessary because this enables an application to say that C_1 extends different unrelated contexts, in the sense that C_1 adds information to both of them, even if the two extended contexts are incompatible; in fact, a View over C_1, which is forced to be consistent, will include only one of the extended contexts
- The implementation of $EXTENDS$ will be triggered to check matching with the three statements, and it will fire the check for knowledge base reorganization
- The check performed by $EXTENDS$ consists of verifying that any View over C_1 that follows the $EXTENDS$ chain does not produce an inconsistent model; therefore, it takes the content of C_1 and of C_2 and runs a DL reasoner (Pellet in this case) over the union. If any inconsistency is detected, $EXTENDS$ tries to isolate the responsible statements, selects those that appear in C_2 and removes them from C_2; the statements are then stored as a Graph G_1. The split is tracked by creating a SplittingReason object, connected to C_2, which is the $source$, and G_1, which is the $result$; it is also connected to a reason, which in this case is instance of the $Inconsistency$ class, and in turn to C_1 which is related as $incompatible$ w.r.t G_1. The statements added to the CRO are reified and attached to the SplittingReason as $triggers$, in order for the split to be traceable, and finally a $part\_of$ relation is asserted between C_2 and G_1. Since the $EXTENDS$ relation is defined transitive, in case C_2 is already connected through a $EXTENDS$ relation to other contexts, then the check is performed not against C_2 alone but over the resulting View; the generated splits in the KB can then be distributed along the $EXTENDS$ chain, which is one of the scalability issues we analyze in Sect. V
- When a View over C_1 is requested, all the CR implementations are requested to provide a set of Contexts or Graphs that *must not* appear in the final view (EXCLUDE set), i.e. are requested to forbid to follow some paths in

the CRO assertions; this is because, when multiple CR are present, some of them may forbid the presence of a result that others would allow to appear in the results; simply removing all the forbidden results after all the paths are followed is not correct nor efficient, since this would require complex pruning strategies. After the EXCLUDE set has been computed, all the CR implementations are required to provide the set of Contexts or Graphs that should appear in the resulting View (INCLUDE set), and they will prune their visiting graph as soon as a forbidden result is reached. The final View is then computed as the INCLUDE set plus the resources connected through $part\_of$ to these elements (not including those in the EXCLUDE set)

- The View can now be viewed as a single model, or the set of URI for the contexts and graphs can be used as dataset for a SPARQL query to be issued to *RDFCore*, which in turn uses ARQ[3] as SPARQL engine to interpret and answer it

V. Results

In this section we presents the empirical evaluation we have conducted so far. In order to check the system for scalability, we needed to design a big knowledge base with non trivial contents, and at the same time divided in smaller chunks without changing the semantics of the content. This, however, seems a very difficult task, and so far we have not found real world ontologies that satisfy these requirements, so we used a homemade tool to generate individuals for a generic ontology; repeating the process many times gave us two well sized knowledge bases.

Using the SOFSEM ontology [4], an ontology to describe the SOFSEM conference, we generated two knowledge bases, one composed of 30 models containing about 70000 statements each (for a total of more than 2 millions triples), and the other containing 900 models of about 2000 triples each (1.8 millions triples); on the first one, we tried to chain the models with $EXTENDS$ relations involving two models at a time, while in the second one we chained tirthy models at a time, obtaining many chains, and then joined the chains. The results are presented in Table I, where the results for the first experiment are presented, in Table II for the second experiment. The second experiment is also depicted in Fig. 6.

As is depicted in the graph, the time elapsed to create a view over the graphs is almost constant, even if the number of relations to navigate increases, while the time elapsed to check the consistency of the models grows proportionally to their size. It is important to note that the consistency check runs only when new relations are enterend in the CRO; the most frequent operation, then, will be the request to create a View starting from some specified models, and the experimental evaluation shows that this operation is usually performed in less than half a second on the test machine (a laptop with 512 MB of RAM, which is not an adequate server setup). The time required to complete the consistency check and automatic splitting on models of greater size is around one minute, which is acceptable from our point of view if we consider that this operation has to be done only once, and occasionally as new relations are added.

[3]http://jena.sourceforge.net

[4]http://nb.vse.cz/~svabo/oaei2006/data/Conference.owl

Model number	Consistency check (ms)	Model number	Consistency check (ms)
0 - 1	63476	10 - 11	60091
2 - 3	49529	12 - 13	62621
4 - 5	54184	14 - 15	59216
6 - 7	58410	16 - 17	58142
8 - 9	62342	18 - 19	62041

TABLE I
Results for 70000 triples models

The most relevant point, here, is that requesting a View operation will return a set of graph identifiers that can be used as dataset for a SPARQL query, ensuring that the model resulting from the union of the queried data is consistent, without having to check at the time of querying; this also means that the memory requirements (at query time) of the framework only depend on the number of relations between Contexts and Graphs, and not on the size of the contained data, or on their complexity. The memory needed by the SPARQL engine to run the query itself, instead, depends heavily on the specific query; still no complete evaluation of the behavior of the system w.r.t. the possible kind of queries has been performed.

VI. Conclusion and Further Works

Basing on the opinion that contexts in Semantic Web KR are a way to tackle some of the current limitations of the languages available and provide for better scalability in some cases, we have presented a theoretical approach and an implementation of Contextual Reasoning in a Semantic Web KB and the associated testing results. We have not only implemented a context mechanism into our KBMS to be able to use a context as a first-class object in assertions, but also illustrated a way to provide for context relations with procedural semantics which – in our opinion – is required for a complete context functionality.

Our next steps will be directed towards the formal definition and implementation of more compatibility relations. Some of them will be as required by the VIKEF project, but we are also interested in exploring more general and domain independent relations between contexts and their properties.

On the implementational side, these planned steps will be accompanied by the development of a more standardized test set and a set of exemplary queries that specifically display and make use of contexts, to assess the practicability, performance and scalability of our implementations.

VII. Acknowledgments

This research was partially funded by the European Commission under the 6th Framework Programme IST Integrated

Model number	View (ms)	Consistency check (ms)	Model number	View (ms)	Consistency check (ms)	Model number	View (ms)	Consistency check (ms)
0	91	2043	9	156	5913	18	208	9986
1	176	2542	10	152	6076	19	214	10950
2	111	2956	11	175	6497	20	218	10699
3	122	3185	12	166	7101	21	232	11434
4	153	3634	13	175	7383	22	267	11696
5	114	3894	14	186	8046	23	242	12064
6	132	4328	15	195	8421	24	244	12706
7	134	4907	16	208	8862	25	249	13095
8	149	5057	17	220	9486	26	276	13476

TABLE II

RESULTS FOR SMALL SIZED MODELS AND LONG CHAINS

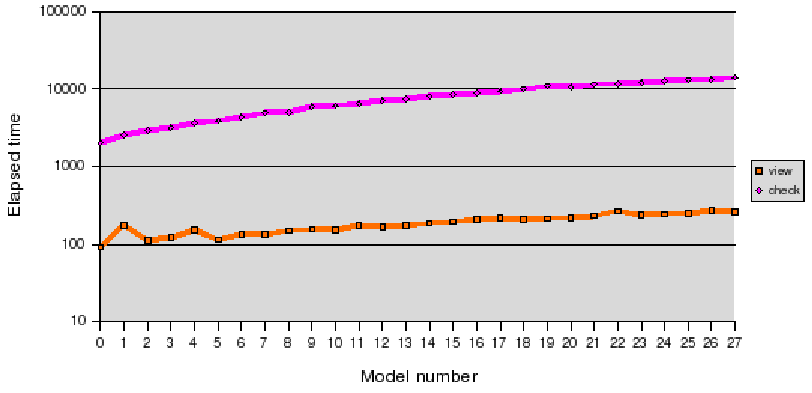


Fig. 6. Results trend for small sized models

Project VIKEF - Virtual Information and Knowledge Environment Framework (Contract no. 507173, Priority 2.3.1.7 Semantic-based Knowledge Systems; more information at `http://www.vikef.net`).

REFERENCES

[1] Massimo Benerecetti, Paolo Bouquet, and Chiara Ghidini. Contextual reasoning distilled. *J. Exp. Theor. Artif. Intell.*, 12(3):279–305, 2000.

[2] Paolo Bouquet, Luciano Serafini, and Heiko Stoermer. Introducing Context into RDF Knowledge Bases. In *Proceedings of SWAP 2005, the 2nd Italian Semantic Web Workshop, Trento, Italy, December 14-16, 2005. CEUR Workshop Proceedings, ISSN 1613-0073, online http://ceur-ws.org/Vol-166/70.pdf*, December 2005.

[3] Jeremy Carroll, Christian Bizer, Patrick Hayes, and Patrick Stickler. Named Graphs, Provenance and Trust. In *Proceedings of the Fourteenth International World Wide Web Conference (WWW2005), Chiba, Japan*, volume 14, pages 613–622, May 2005.

[4] F. Esposito, L. Iannone, I. Palmisano, and G. Semeraro. RDF Core: a Component for Effective Management of RDF Models. In Isabel F. Cruz, Vipul Kashyap, Stefan Decker, and Rainer Eckstein, editors, *Proceedings of SWDB'03, The first International Workshop on Semantic Web and Databases, Co-located with VLDB 2003, Humboldt-Universität, Berlin, Germany, September 7-8, 2003*, 2003.

[5] Chiara Ghidini and Luciano Serafini. Distributed first order logics. In *First International Workshop on Labelled Deduction [LD'98]*, 1998.

[6] Fausto Giunchiglia. Contextual reasoning. *Epistemologia - Special Issue on I Linguaggi e le Macchine*, XVI:345–364, 1993.

[7] Ramanathan V. Guha. *Contexts: A Formalization and Some Applications*. PhD thesis, Stanford, 1991.

[8] Ramanathan V. Guha, Rob McCool, and Richard Fikes. Contexts for the semantic web. In Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *International Semantic Web Conference*, volume 3298 of *Lecture Notes in Computer Science*, pages 32–46. Springer, 2004.

[9] Graham Klyne. *Contexts for RDF Information Modelling*. Content Technologies Ltd, October 2000. http://www.ninebynine.org/RDFNotes/RDFContexts.html.

[10] Graham Klyne. *Circumstance, provenance and partial knowledge - Limiting the scope of RDF assertions*, 2002. http://www.ninebynine.org/RDFNotes/UsingContextsWithRDF.html.

[11] John L. McCarthy. Generality in artificial intelligence. *Commun. ACM*, 30(12):1029–1035, 1987.

[12] John L. McCarthy. Notes on formalizing context. In *IJCAI*, pages 555–562, 1993.

[13] Luciano Serafini and Paolo Bouquet. Comparing formal theories of context in ai. *Artif. Intell.*, 155(1-2):41–67, 2004.

[14] Heiko Stoermer, Ignazio Palmisano, Domenico Redavid, Luigi Iannone, Paolo Bouquet, and Giovanni Semeraro. RDF and Contexts: Use of SPARQL and Named Graphs to Achieve Contextualization. In *Proceedings of the First Jena User's Conference, Bristol, UK*, April 2006. http://jena.hpl.hp.com/juc2006/proceedings/palmisano/paper.pdf.