Networked Graphs: A Declarative Mechanism for SPARQL Rules, SPARQL Views and RDF Data Integration on the Web[*]

Simon Schenk
Institute for Computer Science
University of Koblenz-Landau
Universitätsstraße 1
56070 Koblenz, Germany
sschenk@uni-koblenz.de

Steffen Staab
Institute for Computer Science
University of Koblenz-Landau
Universitätsstraße 1
56070 Koblenz, Germany
staab@uni-koblenz.de

ABSTRACT

Easy reuse and integration of declaratively described information in a distributed setting is one of the main motivations for building the Semantic Web. Despite of this claim, reuse and recombination of RDF data today is mostly done using data replication and procedural code. A simple declarative mechanism for reusing and combining RDF data would help users to generate content for the semantic web. Having such a mechanism, the Semantic Web could better benefit from user generated content, as it is broadly present in the so called Web 2.0, but also from better linkage of existing content.

We propose *Networked Graphs*, which allow users to define RDF graphs both, by extensionally listing content, but also by using views on other graphs. These views can be used to include parts of other graphs, to transform data before including it and to denote rules. The relationships between graphs are described declaratively using SPARQL queries and an extension of the SPARQL semantics. Networked Graphs are easily exchangeable between and interpretable on different computers. Using existing protocols, Networked Graphss can be evaluated in a distributed setting.

Categories and Subject Descriptors

E.1 [**Data**]: Data Structures; E.1 [**Data**]: Data Structures—*Distributed Data Structures*; H.3 [**Information Systems**]: Information Storage and Retrieval

General Terms

Algorithms, Languages

Keywords

Semantic Web, Views, Rules, Distributed Rules, SPARQL, Well Founded Semantics

[*]This research was supported by the European Commission under contract FP6-027026, Knowledge Space of Semantic Inference for Automatic Annotation and Retrieval of Multimedia Content - K-Space and contract IST-2006-027595, Lifecycle Support for Networked Ontologies - NeOn. The expressed content is the view of the authors but not necessarily the view of the K-Space and NeOn projects.

1. INTRODUCTION

Data reuse and integration is the basic idea behind the Semantic Web effort. However, the extremely large and highly distributed setting of the Web makes reuse and integration a difficult task. Although the basic technologies of the semantic web (URIs to denote things, shared ontologies) are well suited for a distributed environment, data reuse and integration today usually takes place in centralized settings. For example, the book mashup[1] and DBPedia[2] connect various sets of semantic data. Although both services are based on semantic web technologies and use freely available data, integration is done by replicating all data to a single repository and connecting data sources using procedural code.

The semantic web suffers from a lack of user generated content and from a lack of services to kick-start wide adoption. One of the reasons for these deficiencies is a missing mechanism for easy, distributed data reuse and integration: As we are talking about quite large datasets in the examples listed above, already the necessary infrastructure for holding the resulting amounts of data keeps many people from offering such services. In order to encourage people to build new services based on existing data, a more flexible and distributed mechanism is needed.

There are two main approaches to data reuse on the semantic web: replication plus procedural code and rule languages. Procedural code is harder to exchange than declarative rules and its semantics is implicit. Replication leads to problems with

1. staleness, as data is frequently updated,
2. scalability, as (combinations of multiple) large datasets can not easily be handled,
3. access rights, as not all data will be available for copying and
4. information exchange, as information sources are connected through programs having implicit semantics, which can not be transferred easily (Webservers do usually not provide the software used along with the content they deliver.).

Rule languages such as the Semantic Web Rule Language SWRL [10], N3 [3] and the upcoming Rule Interchange For-

[1]http://sites.wiwiss.fu-berlin.de/suhl/bizer/bookmashup/
[2]http://dbpedia.org/

mat RIF [9] allow for defining how to reuse data in a declarative way. However, they work best in a centralized setting, because apart from N3, none of them allows users to specify where the data to reason on comes from and which dataset the result should belong to. Additionally, these languages are unfamiliar for people without a background in logics — and most people using semantic web technologies now and in the future can be assumed not to have such a background. Finally, they have not been designed to gracefully handle recursion and negation in a dynamic environment like the web. As we will see in our use case, these properties are already desirable for simple scenarios.

We propose Networked Graphs as a declarative mechanism to define RDF graphs both extensionally, by listing statements, and intensionally using views on other graphs. SPARQL and a small syntactic extension to RDF are used to define views. The proposed mechanism is very powerful and flexible, as it allows one to use almost all of the expressiveness of SPARQL CONSTRUCT queries, including negation, and — when used inside Networked Graphs — recursive views. Networked Graphs are designed to be evaluated in a distributed setting and are easily exchangeable. Users of Networked Graphs do not need to learn a new rule language, but can use the standard language SPARQL — which we expect to be widely adopted soon.

Contribution

In this paper we define a syntactic extension to RDF which allows for expressing views in RDF graphs in a declarative way in section 4. We define the semantics of NGs through an extension of the SPARQL semantics in section 5. We describe a prototypical implementation in section 6 and finally shortly discuss the complexity of reasoning with NGs is in section 7. Foundations and extensions are illustrated using a running example. We start our description in the following Section 2 with a use case from which we derive requirements, and which serves as a running example in the remainder of the paper. We introducw existing formalisms in Section 3. In the following we use "*NG*" to abbreviate "*Networked Graph*".

2. USE CASE

Mike is project manager of a new semantic web research project. As with most new projects, he needs to build a project website containing information about the people working in the project. We talk about a Semantic Web project, so all information shall be available in RDF. All project members are Semantic Web enthusiasts and provide information about themselves in their personal foaf[3] files on the web. Mike would like to avoid replicating this information. He also does not want to have to update the project website, whenever one of the members changes her personal information. Mike thinks a view mechanism for RDF would be ideal to extract the information he needs.

Reusing Information. Mike's first task is to build a page containing contact information of all project members. He sets up a graph mikesProject. He defines a view feeding into mikesProject based on the foaf files of the project members. It imports names, e-mail addresses, phone numbers and employers of all people working in the project.

[3]http://www.foaf-project.org/

Importing third-party data. After a couple of months, Mike is asked to include all publications done within the project to mikesProject. All project members work full time for the project. Therefore Mike can assume that all their publications done during the lifetime of the project are relevant. He adds a view based on the DBLP RDF mapping[4], including all publications by project members listed in mikesProject within the time period of 2007-2009. Whenever anything relevant is newly published, it will be included in the project website automatically.

Negation. Mike thinks, it will be nice, if the project acknowledges all those authors who have contributed to publications done in the project, but who do not work in the project themselves. He sets up a new view, adding acknowledgements to all those authors of papers, which have creators from the project, but where the persons acknowledged are not listed as project members in mikesProject. Mike uses the Semantic Web query language SPARQL, which allows to formalize this rule using negation by failure.

Recursion. Bob works in the project. He finds the information on the project website very useful. As the project is rather small, he assumes he knows everybody in the project. For this reason he adds a view to his foaf file bobFOAF, which creates a foaf:knows relation between Bob and every person listed as a project member in mikesProject. This results in a circular dependency between bobFOAF and mikesProject.

Exchange of NGs. Mandy manages a similar project. Today she is faced with the task of creating a new project website. Mandy likes Mike's website a lot and would like to set up something similar. Usually, her semantic web browser immediately computes all views. She did not even notice that Mike uses them. Today, however, she looks at mikesProject with a text editor and sees all the view definitions. She replaces the datasets of the views such that the foaf files of her own project's members are used. After only five minutes of work she has created a rich project description in graph alicesProject.

Requirements

We observe the following requirements from the presented use case:

1. RDF graphs shall be expressable directly, like DBLP in our use case, and/or through views on other graphs, like mikesProject of bobFOAF, which is a mixture of both.

2. In a dynamic and distributed setting, like the Semantic Web, it is difficult to avoid circular dependencies among NGs, which therefore must be supported by the semantics.

3. View definitions must support negation, as available in SPARQL and as used here for the definition of acknowledgements.

4. Definitions of graphs should be easily exchangeable and, hence, must be self-describing, i.e. include all necessary view definitions.

5. The extension needs to be upward compatible, i.e. it should be based on existing Semantic Web building

[4]http://www4.wiwiss.fu-berlin.de/dblp/

blocks. Downwards, it should at least not prevent existing RDF technology from working in the foreseen, standardized way.

Further Possibilities for Applications

Apart from the rather simple use case we will use throughout this paper, we have recognized several other application areas, where NGs, will be extremely useful for describing rules, views and data integration.

Linking Dataspaces. A new line of research in databases aims at creating and linking dataspaces [6], i.e. heterogeneous spaces of information, such as available in small databases and on PC desktops. While the gist of this work is related to our objectives, they pursue their goals with idiosyncratic mechanisms rather than by standardized languages as we do with our minor extension of RDF and SPARQL towards NGs.

Distributed Rules for Policies. Alves et al. propose a distributed, rule based policy mechanism [2]. Their language extends Prolog in a proprietary way. Using NGs an elegant integration of the policy mechanism with the semantic web could be achieved.

Decentralized Data Integration. DBPedia integrates various datasets, which are available in RDF. Among others, links to the geonames ontology[5] and to DBLP are included. An aggregated dataset is published in irregular intervals. This data could directly be integrated using networked graphs, immediately including changes from any of these datasets.

3. FOUNDATIONS

In this section we briefly describe existing foundations of NGs.

3.1 RDF

RDF is a graph based knowledge representation language. The nodes in a graph are URIs, blank nodes (a kind of existentially quantified variables) or literals. Arcs between the nodes, labeled with URIs, represent their relationships. We simplify the RDF graph model [14] here slightly in order to come up with a more concise formal characterization. Analogous to the SPARQL specification [4] we allow literals as subjects of statements.

Definition 1 (RDF Graph).
Let U be the set of URIs, L the set of RDF Literals and B the set of Blank Nodes as defined in [14]. U, L and B are pairwise disjoint. Let $R = U \cup L \cup B$. A statement *is a triple in $R \times U \times R$. If $S = (s, p, o)$ is a statement, s is called the subject, p the predicate and o the object of S. An RDF* graph *G is a set of statements. For every two RDF graphs G_1 and G_2 the sets of blank nodes used in G_1 and in G_2 are disjoint.*

[14] defines a model theoretic semantics for RDF graphs and the RDF Schema Language RDFS. RDFS includes, for example, transitivity of class subsumption, the instanceOf relationship and others. We refer the reader to [14] for a detailed description of RDF and the RDF Schema language RDFS.

[5]http://www.geonames.org/

3.2 Named Graphs

While the RDF recommendation does not allow referring to whole RDF graphs, named graphs introduced in [11] offer means to group a set of statements in a graph and to refer to this graph using a URI. This way information about the graph can be expressed in RDF using its name as a subject or object:

Definition 2 (Named Graph).
A named graph is a pair (n, G) of a URI n, called the name, *and an RDF graph G, called the* extension.

The following example illustrates the triple structure of RDF and the concept of named graphs using Trig notation[6] The statements in a named graph are listed in curly brackets, prepended by the name of the graph. Due to space constraints we will use abbreviated URIs without namespaces.

Example 1.

```
:DBLP {:NGPaper dc:creator :Alice.
       :NGPaper dc:creator :Bob. }

:bobFOAF {
  :bobFOAF foaf:primaryTopic   :Bob.
  :Bob     foaf:name           "Bob"^^xsd:String.
  :Bob     foaf:currentProject :K-Space.
  :K-Space foaf:fundedBy       :EU.
  :Bob     foaf:phone          "+49-261-287-2868".}

:MikesProject {
  :SemWebProj foaf:name "Networked Graphs".}
```

3.3 SPARQL

SPARQL is a query language for RDF based on graph pattern matching, which is defined in [4]. In this paper we are only interested in SPARQL CONSTRUCT queries. A CONSTRUCT query matches a graph pattern against one or more input graphs. The resulting variable bindings are embedded into a graph template in order to generate new RDF data. To further restrict the bindings produced by pattern matching, filter expressions can be used, for example to check for (in-)equality of variables, datatypes of literals etc.

A SPARQL query is evaluated against a *dataset* consisting of a set of named graphs (declared using FROM NAMED clauses) and a default graph, which is the union of one or more named graphs (declared using FROM clauses).

A query returning `foaf:currentProject` triples for all persons with some foaf document in the dataset is shown in example 2. It selects all persons from all foaf files Mike uses to built mikesProject. Additionally, their phone numbers are included, if available.

Example 2.

```
CONSTRUCT {?member foaf:currentProject ?project.
           ?member foaf:phone ?phone}
FROM NAMED :bobFOAF FROM NAMED :chrisFOAF FROM NAMED...
WHERE {
 GRAPH ?foafFile {
  ?foafFile foaf:primaryTopic ?member
  OPTIONAL { ?member foaf:phone ?phone } } }
```

SPARQL can be used to model negation by failure using a combination of OPTIONAL graph patterns and the BOUND filter, which checks whether a variable has been bound during graph pattern matching. We will call such a construct in a query *bound negation.* Example 3 illustrates

[6]http://www.wiwiss.fu-berlin.de/suhl/bizer/TriG/Spec/

the use of bound negation to select co-authors of project members, whom Mike wants to thank for contributing to papers. In this query we first find all co-authors of project members. We then try to find an (optional) `foaf:currentProject` relation between the project and the co-authors. If this fails (i.e. the co-author is not a project member), the variable `?x` is left unbound, and the filter expression at the end of the query succeeds. The obtained variable bindings are embedded in the CONSTRUCT template to add an acknowledgement to the co-author[7].

EXAMPLE 3.

```
CONSTRUCT {:SemWebProject :acknowledges ?author}
FROM NAMED :DBLP        FROM NAMED :mikesProject
WHERE {
 GRAPH :DBLP {
  ?paper dc:creator ?author.
  ?paper dc:creator ?member }.
 GRAPH :mikesProject {
  ?member foaf:currentProject :SemWebProject
  OPTIONAL { ?x foaf:currentProject :SemWebProject
             FILTER (?x = ?author) }
  FILTER (!BOUND(?x)) } }
```

4. INTRODUCING NETWORKED GRAPHS

We propose *NGs* to fulfil the requirements listed in section 2 by extending the technologies listed in the previous section. To address requirement (1) on expressing RDF graphs by extensional and intensional means, we extend RDF with a SPARQL based view mechanism. To support exchangeability and backwards compatibility (requirements 4 and 5), we define a syntax based on named graphs. By this extension it becomes possible to define the meaning of such a *network* of graphs by the way they reference each other. To deal with negation and cycles (requirements 2 and 3), we adapt a suitable semantics from logic programming in section 5 and explain how it can be computed in a distributed setting in section 6.

In this section we start by defining NGs using an abstract syntax, before we introduce the RDF and SPARQL based syntax for NGs.

4.1 Abstract Syntax

In the previous section we have seen how SPARQL can be used to express a single view in a dataset. What is still missing is a link between the view definition and the graph, which shall include the view. This link is given be NGs:

DEFINITION 3 (NETWORKED GRAPH).
A NG is a quadruple $G^W = (n, G, \left[G_1^W, ..., G_n^W\right], G^W)$, where n is a URI called the name *of the NG, G is an RDF graph, $\left[G_1^W, ..., G_n^W\right]$ is a list of NGs and v is a mapping from a list of NGs to an RDF graph called the* view definition *of G^W.*

We define a function *deref* to access the contents of a NG given a NG definition. *deref* carries out dereferencing of NGs as a logical operation, i.e. given a NG it computes the content of the NG. This logical operation is opposed to dereferencing as an addressing operation, i.e. given a URI, accessing the corresponding file or SPARQL endpoint. In contrast to named graphs, NGs are not necessarily listed in a single file or at a single endpoint, as parts of them are inferred. In Section 5 we will describe, how *deref* can be computed for a set of NGs.

DEFINITION 4 (DEREF).
Let $V = (n, G, \left[G_1^W \ldots G_n^W\right], v)$ be a NG.
Then deref *is a function mapping from networked graphs into a set of statements, i.e. a graph:*
$deref(G^W) = G \cup v(deref(G_1^W), ..., deref(G_n^W))$

From these definitions it is clear, that named graphs are a special case of NGs with $G^W = (n, G, [], \emptyset)$. In our use case scenario, the graph mikesProject could be formalized as follows:

EXAMPLE 4.
Let bobFOAF, chrisFOAF and DBLP be NGs. Then mikesProject is a NG:
mikesProject = (:mikesProject,
  {(:SemWebProj foaf:name "Networked Graphs")},
  [bobFOAF, chrisFOAF, DBLP, mikesProject], v),
with v(bobFOAF, chrisFOAF, DBLP, mikesProject) =
  {(:Bob foaf:currentProject :SemWebProject),
  (:Bob foaf:phone "+49-261-287-2868"),
  (:Chris foaf:currentProject :SemWebProject),
  (:SemWebProject :acknowledges :Alice)}

For reasons of simplicity, we will use the name of a NG interchangeably with the NG itself. From the context it will be clear, whether the name or the NG is meant.

4.2 An RDF Language Extension for Networked Graphs

We now provide an RDF syntax for NGs based on named graphs for the extensional definitions and SPARQL for the intensional definitions.

DEFINITION 5 (NETWORKED GRAPHS SYNTAX).
A NG $G^W = (n,G,[G_1^W,...,G_n^W],v)$ is defined *in a named graph with name* **n** *and extension G. The view definition is included in statements of the form:*
***n** g:definedBy <query>.*[8]
where <query> is a literal containing a CONSTRUCT query. The datatype of <query> is g:query. We call <query> a subquery of v. We call such a statement a view definition statement. In general, a named graph may contain a number m of such view definition statements. The overall view is then defined by $v(G_1^W, \ldots, G_n^W) = \bigcup_{i=1}^{m} v_i(G_1^W, \ldots, G_n^W)$, where v_i is the evaluation function of the i-th subquery.

`definedBy` statements are intended to carry two meanings. First, there is the extensional meaning that the statements exists. Second, there is the intensional meaning, i.e. the evaluation of the view it defines. The second meaning will only be incurred, if the graph in which a `definedBy` statement occurs is identical with the subject of this statement. Existing machinery for working with named graphs can be used for serialization and exchange of NGs. A non NG aware repository can still interpret the extensionally

[7]Using negation in SPARQL puts a heavy burden on the user. As negation is available in SPARQL, it should be available in an easily understandable way — unlike it is specified today. We do not target at new language constructs for negation in SPARQL here, but we must still show how to deal with the existing specification. Thereby, negation adds substantially to the complexity of the formalism. A future improvement of SPARQL in this direction would be beneficial for the syntactic clarity of Networked Graphs.

[8]As namespace for the NG vocabulary we propose http://isweb.uni-koblenz.de/ontologies/2006/11/ng#

listed statements in a NG, providing upwards compatibility.

Using this syntax, the definitions of bobFOAF and mikesProject are listed in example 5. In both cases, a named graph holds the extensions introduced in example 1 and the view definitions. In bobFOAF, `foaf:knows` relations are created from Bob to all project members listed in mikesProject, except for Bob himself. mikesProject contains view definitions encoded in three queries. The first one adds project members and their phone numbers, if available. The second one analogously imports publications from DBLP, so we only sketch it here. The third one adds acknowledgements to all authors listed in DBLP, such that one of their publications has an author, who is listed as project member in mikesProject, but the acknowledged person herself is not a project member, as explained in example 3.

Example 5.

```
:bobFOAF {
 :bobFOAF foaf:primaryTopic   :Bob.
 :Bob     foaf:name           "Bob"^^xsd:String.
 :Bob     foaf:currentProject :K-Space.
 :K-Space foaf:fundedBy       :EU.
 :Bob     foaf:phone          "+49-261-287-2868".
 :bobFOAF g:definedBy
   "CONSTRUCT {:Bob foaf:knows ?person}
    FROM :mikesProject
    WHERE { ?person foaf:currentProject :SemWebProject
            FILTER (?person != :Bob) }"^^g:query.
}

:mikesProject {
 :SemWebProj foaf:name "Networked Graphs".
 :mikesProject g:definedBy
   "CONSTRUCT {?member foaf:currentProject ?project.
              ?member foaf:phone ?phone}
    FROM NAMED :bobFOAF    FROM NAMED :chrisFOAF ...
    WHERE { GRAPH ?foafFile {
      ?foafFile foaf:primaryTopic ?member
      OPTIONAL {?member foaf:phone ?phone}}}"^^g:query.
 :mikesProject g:definedBy
   "CONSTRUCT {?paper dc:creator ?author}..."^^g:query.
 :mikesProject g:definedBy
   "CONSTRUCT {:SemWebProject :acknowledges ?author}
    FROM NAMED :DBLP       FROM NAMED :mikesProject
    WHERE {
      GRAPH :DBLP { ?paper dc:creator ?author.
                    ?paper dc:creator ?member}.
      GRAPH :mikesProject {
        ?member foaf:currentProject :SemWebProject
        OPTIONAL { ?x foaf:currentProject :SemWebProject
                   FILTER (?x = ?author) }
        FILTER (!BOUND(?x)) } }"^^g:query.
}
```

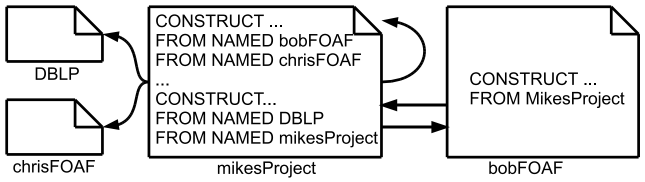

Figure 1: An Interdependence Set

Figure 1 illustrates the interdependencies of NGs resulting from the definitions in example 5. We call such a set of interdependent graphs an *interdependence set.*

5. SEMANTICS

We now revisit the requirements from our use case to discuss requirements on the NGs semantics. Afterwards we give an overview of the semantics. A detailed definition is given in the appendix. For the understanding of the detailed semantics, some knowledge of the SPARQL algebra [4] and of the well founded semantics for logic programs [20] is required. Unfortunately we do not have enough space to repeat the these foundations in detail here.

5.1 Requirements on the Semantics of NGs

As we have pointed out when describing the use case, we envision NGs to be a mechanism used by a broad range of agents on the semantic web to recombine RDF data. In this environment, it is not possible to pose restrictions on the use of NGs (requirement 2), for example to ensure hierarchical dependencies between NGs or stratified use of negation (i.e. no view using negation directly or indirectly depends on itself).

Cyclic Definitions of Networked Graphs. A scenario that may lead to problems with naive interpretations is given when you assume that Mikes assistant Anna persuades all acknowledged authors to join the project. She adds a view to mikesProject saying that everybody acknowledged has `:SemWebProject` as a `foaf:currentProject`. Now obviously nothing can be inferred about acknowledged authors anymore, because they are only acknowledged if they are *not* members of the project. However, we still want to have contact information and literature of everybody else available on the website, as they are not affected by this contradiction. The semantics of NGs must be able to handle such mutual dependencies via negated premises.

SPARQL, Rules and Logic Programs. Networked Graphs are quite similar to logic programs. In fact we show how SPARQL can be mapped to acyclic Datalog with negation in [17]. Independently, Polleres defines a mapping to logic programs under the stable model semantics [16]. We define the semantics of NGs analogously to a fixpoint semantics for logic programs. From a high level point of view we treat SPARQL CONSTRUCT queries as rules with negation. While the encapsulation of SPARQL algebra expressions into a fixpoint semantics may seem a bit complicated at first, it has two clear advantages: First, it allows for using existing SPARQL endpoints. Second, NGs can automatically benefit from future extensions of SPARQL such as aggregates or user defined filter functions.

RDF Schema. SPARQL defines *entailment regimes* under which a query can be evaluated. The standard entailment regime is simple RDF entailment. However, more complex regimes like RDFS are possible, the only requirement being monotonicity of the entailment regime. Using a more complex entailment regime, NGs can be evaluated taking RDFS into account, without additional extensions to NGs. (Note that NGs are expressive enough to alternatively model the RDFS inference rules as view definitions.) Sirin and Parsia propose an entailment regime for a subset of SPARQL, which can be used to query OWL ontologies taking into account OWL DL semantics [5]. Using such an entailment regime could even bring the power of OWL to a subset of NGs. Interaction with OWL, however, would need a detailed investigation of the resulting combined language. In the following we assume a basic RDF entailment regime.

5.2 High-Level View of the Semantics of Networked Graphs

We address the requirements on the semantics, especially regarding reasoning in the presence of contradictions, by adapting the well founded semantics (WFS) for logic programs [20] to NGs. In this section we describe the core idea of the WFS. Details are given in the appendix.

Fixpoint Model for Well-founded Semantics. The WFS is based on a three valued logic with truth values of `true`, `false` and `unknown`. It assigns truth values of true and false to as many facts as possible. For facts, where such an assignment is not possible, the truth value assigned is unknown. This is the case for non-stratifiable negation as sketched above. The WFS treats true and false facts symmetrically, and is defined using the fixpoint of a direct consequence operator T, which we briefly explain in the following paragraphs.

The WFS for NGs assumes all extensionally listed statements and everything we can follow from these statements without using bound negation to be `true` and by assuming all statements in the so called *greatest unfounded set* to be `false`. To explain the unfounded set, we need the notion of *ground instantiated views*. A view is ground instantiated by replacing every variable in the view with an element of the set of RDF Resources R. Hence, the statement patterns in a ground instantiated view do not contain variables any more.

We say an optional statement used in bound negation like `(?x foaf:currentProject :SemWebProject)` in example 3 is *matched negatively*. All other statements are matched positively. A statement S is in an unfounded set U, if for all ground instantiated views v with S in the CONSTRUCT pattern one of the following holds: (1) All negatively matched statements in v are known to be true or (2) Some positively matched statement in v is known to be false or (3) Some positively matched statement in v is in U. The greatest unfounded set is the union of all unfounded sets.

The third condition applies to mutually dependent ground instantiated views, where none of the views can be established to be true or false first.

The sets of true and false statements are extended iteratively by computing the direct consequences of the known true and false statements until a fixpoint is reached. The direct true consequences are statements in the CONSTRUCT patterns of ground instantiated views, which match the known true and false statements from the previous iteration. A new unfounded set based on the knowledge from the previous iteration is determined. All statements in the new unfounded set are direct false consequences.

Statements that are not assigned a value of true or false, when the fixpoint is reached, are assigned a truth value of unknown statements. We define the semantics of a NG G^W, $deref(G^W)$, as the set of true statements in the graph when reaching the fixpoint.

Example. In our example, after Anna has added her new view, all ground instantiations of this new view together with the view defining the acknowledgements form an unfounded set: every statement of the form `(:semWebProject :acknowledges sbd)` negatively depends on a statement of the form `(sbd foaf:currentProject :SemWebProject)`, which in turn depends on the first statement. Hence, no statements about acknowledged authors are inferred. However, all views, which are not directly affected, are not influenced by the contradiction, because the statements they generate are not part of an unfounded set.

Procedural Semantics. Obviously it is difficult to work with all possible ground instantiated statements and to guess an unfounded set, because of combinatorial explosion and non-determinism. A procedural semantics is available, however, which we shortly explain in the next section.

6. IMPLEMENTATION

We have implemented a NG reasoner as an extension for the Sesame 2 RDF repository. Implementation, public prototype and test data are available at `http://isweb.uni-koblenz.de/Research/NetworkedGraphs`.

The implementation uses a variation of the alternating fixpoint algorithm for computing the WFS [7], which does not require to guess an unfounded set. The algorithm is initiated with true statements in an interdependence set. It then alternates between an overestimation of the set of true, inferred statements and an underestimation of this set. Eventually, this alternation converges to the set of true statements in the interdependency set.

In some more details, the procedure starts with the true statements, which are extensionally listed, or which can be derived from views, which do not use negation. We call this underestimate U_1. Statements in U_1 are known to be true. U_1 is used to compute an overestimate O_1 by evaluating all views against this set of true statements. The result will be an overestimate, because U_1 was still incomplete and therefore bound negation will succeed in too many cases. O_1 represents the set of *potentially* true statements. We do not know for every statement in O_1, whether it really is true, but at least it is not definitely false. Now we compute a new underestimate, using O_1 to match negatively matched statements and U_1 to match all positively matched statements. The result U_2 is an underestimate of true statements that is larger than U_1.

When after some iterations we come up with the same under- and overestimates as in the previous iteration, a fixpoint has been reached. The underestimate then equals the true statements under the WFS, i.e. it represents the semantics of the interdependence set.

The algorithm is combined with tabling to reduce the amount of redundant work. Tabling means we save intermediary results during the iterative computation, instead of recomputing them when needed. Apart from saving redundant work, this also reduces network traffic during distributed evaluation. Additionally, we apply heuristics to filter out irrelevant parts of interdependence sets. NGs evaluation takes place at query time in the current prototype.

6.1 Initial Experiments

First tests have shown that the prototype works well with medium sized datasets of up to 200,000 statements. Queries against sets of 20 to 60 NGs with 20 to 60 views could be answered within 3 to 120 seconds (depending on the size and complexity of the problem) on a 2GHz Pentium M. A scenario like our use case can be evaluated in three iterations. To improve scaleability, we will improve integration with the Sesame query engine and the pruning of the search space in future work.

6.2 Distributed Evaluation

Every view, together with its table of interim results and some status information used for the detection of cycles, is represented as a single 'QueryNode' object. This allows to distribute NG reasoning by distributing QueryNodes to the machines holding the relevant graphs. The communication between QueryNodes is done using the existing SPARQL protocol, with small extensions for control data. The extensions are threefold:

First, when involving multiple endpoints in NG evaluation, we need to detect and avoid cycles. Hence, we extend the SPARQL query protocol with a list of queryIDs. Whenever a QueryNode forwards a query to evaluate some view, it generates a new queryID and appends it to the list. Upon receiving a query, a QueryNode compares the list of queryIDs in the query message with the list of the queryIDs it has generated itself. If a cycle is detected, subsequent queries are not forwarded to other peers. The received query is only evaluated on local data and the result is returned. Using this extension, an iteration of the alternating fixpoint procedure is started by a single QueryNode and triggered on other nodes in a tree shaped hierarchy. After every iteration, every QueryNode updates its table of results. Only new results are returned to other nodes.

Second, through indirect dependencies it may happen that the same QueryNode is triggered twice in the same iteration. To avoid duplicate work or errors in the tables, each queryID is augmented by an iteration number. If an already computed combination of queryID and iteration number is received, the query is answered from the local table.

Finally, the query result message is extended with a flag signalling that the answering node considers the evaluation of its view done – either because all views it depends on are also done, or because it is the root of a subtree of the QueryNode network, for which a fixpoint has been reached. This last extension is only used to improve performance, as it can save the last iteration of the alternating fixpoint algorithm. The fixpoint could also be detected by the node which started the evaluation.

The protocol is defined as an extension to the SPARQL protocol specification[9] and available from the implementation webpage. As the mechanism is based on existing standards, it allows the usage of existing SPARQL endpoints when defining NGs. When dealing with non NG aware repositories, one may fall back to standard SPARQL.

7. COMPLEXITY

We briefly investigate the data complexity of NG evaluation. The data complexity is defined as the complexity of computing a model for a variable extensional knowledge base given a fixed query. Data complexity is of particular interest here, as usually the view definitions are short compared to the knowledge base. Further we expect view definitions to be rather stable, while the extensionally defined statements might frequently change.

If blank nodes are created in the CONSTRUCT patterns of views and these views are used recursively, evaluating NGs becomes undecidable in general. In this case, we could have infinitely many ground instances of statement patterns. For this reason we limit the discussion to NGs without blank node creation.

THEOREM 1. *The data complexity of NG evaluation without blank node creation is quadratic in the size of the NGs in the interdependence set.* □

If negation is not used in recursive view definitions, we can give a tighter bound:

[9]http://www.w3.org/TR/rdf-sparql-protocol/

THEOREM 2. *The data complexity of NG evaluation without value creation and with stratifiable negation is linear in the size of the NGs in the interdependence set.* □

The combined complexity is the complexity of computing a model for both variable queries and knowledge bases.

THEOREM 3. *The combined complexity of NG evaluation without value creation is in EXPTIME.* □

All proofs are based on mappings to fragments of logic programming and their well known complexity. Proofs for theorems 1 and 2 and can be found in [18]. The proof of theorem 3 is derived from the facts that NGs can be mapped to datalog with well founded negation [18], which has EXPTIME combined complexity (implicit in [20]). NGs creating blank nodes correspond to logic programs with function symbols, which are undecidable in general under the WFS.

8. RELATED WORK

Related work comes from several areas, especially from rules and views for the semantic web.

OWL Contextualization. For the Web Ontology Language OWL, which is based on RDF, declarative approaches for import and contextualisaton have been proposed: `owl:imports` is a mechanism for OWL which includes a *whole* OWL file in another one. It is a rather coarse mechanism, not flexible enough for the use case presented here. C-OWL [13] is an extension of OWL with local contexts connected with bridge rules. C-OWL aims at translations between local contexts, not at reuse of RDF/OWL data.

Rules to deal with (variations of) named graphs. Approaches for dealing with RDF graphs with either a declarative or a procedural semantics include the following:

Stoermer et al. use named graphs plus semantic extensions of RDF to implement a system using modal logic for statements in RDF graphs [8]. The RDF semantics is extended, such that statements are only true in the context of a graph called a context. Between these graphs compatibility relations can be defined and inferred. The focus lies on describing relationships among graphs. Reuse of RDF data, for example by importing statements, is not possible.

The NEPOMUK Representation Language NRL [19] is used in the context of a Social Semantic Desktop to allow for a kind of views and to apply various different semantics to RDF graphs. NRL allows for various ways to specify "views", which are defined with a *procedural* semantics in contrast to the *declarative* semantics of NGs. Applied in a heterogeneous and distributed web environment, we expect problems with this procedural semantics, because implications will become hard to understand.

N3Logic [3] is a rule language built around RDF. Its expressive power is comparable to NGs for positive rules. Only a weaker form of negation is supported by N3Logic. N3Logic defines several builtins, e.g. cryptographic functions, which are not available in NGs, but could be modeled using user defined filter functions in SPARQL. An advantage of NGs in our opinion is its foundation on SPARQL — both to save users the effort of learning a new language and to directly benefit from SPARQL implementations and extensions.

Views. [21] and [12] propose declarative view mechanisms for RDF. They can be used to define "virtual" classes, properties and instances based on graph patterns. RDFS semantics is employed to ensure that necessary class relations are also included in the view. The purpose is to provide a view to a graph using a different ontology rather than defining graph contents based on other graphs. These classical views are not oriented towards reuse and exchange of views across the borders of single RDF repositories.

Semantic Web Pipes[10] can be used to define RDF documents as views on other documents. An XML syntax is used and cyclic dependencies are not supported. Semantic Web Pipes are designed rather as a web service than as a declarative mechanism.

ActiveXML Abiteboul et al. describe ActiveXML documents [1], which represent a concept very similar to NGs, but based on XML. ActiveXML documents may contain intensional subtrees defined through web service calls or XPath queries. The semantics of ActiveXML documents is also defined using a fixpoint, but unlike NGs, ActiveXML documents are infinite in general. In contrast to ActiveXML, NGs support a powerful kind of negation. While NGs can support various entailment regimes through the use of SPARQL, ActiveXML documents have weaker semantics based on XML.

9. CONCLUSION

We have introduced Networked Graphs as a means for describing RDF graphs that are partially derived from other graphs using a declarative view mechanism. We base Networked Graphs on work on SPARQL and Named Graphs. Extensions allow for recursive view definitions, i.e. a kind of rules, and easy exchange of self describing Networked Graphs. Thus, Networked Graphs allow for defining, exchanging and executing SPARQL rules, SPARQL views and RDF data integration in a decentralized fashion. We have shown that the complexity of our extension allows for use at web scale, especially if negation is used carefully. Thus, we propose a mechanism for easy information reuse and recombination and have fulfilled the requirements derived from our running use case — which is just one of the many cases that will benefit from dynamic networking between RDF graphs.

10. REFERENCES

[1] S. Abiteboul, O. Benjelloun, and T. Milo. Positive Active XML. In *Proc. of PODS-2004*, 2004.

[2] M. Alves, C. V. Damásio, W. Nejdl, and D. Olmedilla. A Distributed Tabling Algorithm for Rule Based Policy Systems. In *Proc. of POLICY-2006*, 2006.

[3] T. Berners-Lee, D. Connolly, L. Kagal, Y. Scharf, and J. Hendler. N3logic: A logic for the web. *J. of Theory and Practice of Logic Programming (TPLP), Special Issue on Logic Programming and the Web*, 2007.

[4] E. Prud'hommeaux, A. Seaborne (eds.). SPARQL Query Language for RDF. http://www.w3.org/TR/rdf-sparql-query/, 2007.

[5] E. Sirin and B. Parsia. SPARQL-DL: SPARQL Query for OWL-DL. In *Proc. of OWLED 2007*, 2007.

[6] M. J. Franklin, A. Y. Halevy, and D. Maier. From Databases to Dataspaces: A new Abstraction for Information Management. *SIGMOD Record*, 34(4), 2005.

[7] A. V. Gelder. The alternating fixpoint of logic programs with negation. In *Proc. of ACM SIGACT-SIGMOD-SIGART-1989*, 1989.

[8] H. Stoermer et al. RDF and Contexts: Use of SPARQL and Named Graphs to Achieve Contextualization. In *Proc. of 2006 Jena User Conference*, 2006.

[9] Harold Boley and Michael Kifer (eds.). RIF Core Design. http://www.w3.org/TR/rif-core/, 2007.

[10] I. Horrocks et al. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. http://www.w3.org/Submission/SWRL/, 2004.

[11] J. Carroll et al. Named Graphs, Provenance and Trust. In *Proc of WWW-2005*, 2005.

[12] A. Magkanaraki, V. Tannen, V. Christophides, and D. Plexousakis. Viewing the semantic web through RVL lenses. *J. Web Semantics*, 1(4), 2004.

[13] P. Bouquet et al. Contextualizing Ontologies. *J. of Web Semantics*, 1(4), 2004.

[14] P. Hayes (ed.). Rdf semantics. http://www.w3.org/TR/rdf-mt/, 2004.

[15] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and Complexity of SPARQL. In *Proc. of ISWC-2006*, 2006.

[16] A. Polleres. From SPARQL to rules (and back). In *Proc. of WWW-2007*, 2007.

[17] S. Schenk. A SPARQL Semantics Based on Datalog. In *Proc. of KI2007*, 2007.

[18] S. Schenk and S. Staab. Networked RDF Graphs. Technical report, 2007. http://uni-koblenz.de/~sschenk/publications/2007/ngtr.pdf .

[19] M. Sintek, L. van Elst, S. Scerri, and S. Handschuh. Distributed Knowledge Representation on the Social Semantic Desktop: Named Graphs, Views and Roles in NRL. In *Proc. of ESWC-2007*, 2007.

[20] A. van Gelder, K. Ross, and J. S. Schlipf. The Well-Founded Semantics for General Logic Programs. *J. of the ACM*, 38(3), 1991.

[21] R. Volz, D. Oberle, and R. Studer. Implementing Views for Light-Weight Web Ontologies. In *7th International Database Engineering and Applications Symposium.* IEEE, 2003.

[10]http://pipes.deri.org/

APPENDIX

A. FORMAL SEMANTICS OF NETWORKED GRAPHS

We limit our definitions here to well defined graph patterns as defined in [15]. Graph patterns, which are not well defined introduce another kind of non monotonicity apart from *bound negation*, which would complicate the following definitions.

Definition 6 (Well Designed Graph Pattern). *A graph pattern P is* well designed*, if for every occurrence of a sub-pattern $P_0 = (P_1\ OPTIONAL\ P_2)$ of P and for every variable $?x$ occurring in P, the following condition holds: if $?x$ occurs both inside P_2 and outside P_0, then it also occurs in P_1.*

Further, we forbid the use of result modifiers: ORDER does not make sense as the result of a CONSTRUCT query is a graph and as such a set. When LIMIT and OFFSET are used to retrieve only a slice of the query results, the semantics of a query may become unclear, as the result becomes implementation specific.

Data Structures	
U	The set of all URIs
R	The set of all URIs, blank bodes and RDF literals
V	The set of all SPARQL variables
S	Polarized statement, a quadruple of the form $R \times U \times R \times \{true, false\}$
G^W	Well founded graph, a set of polarized statements
Q	A well formed SPARQL query
D	Dataset, a set of the form $\{G, (n_1, G_1), (n_2, G_2), ...(n_n, G_n)\}$
I	Set of interdependent graphs

Auxiliary Functions	
$pos(G^W)$	$\{(s,p,o) \mid (s,p,o,+) \in G^W\}$
$neg(G^W)$	$\{(s,p,o) \mid (s,p,o,-) \in G^W\}$
$dSet(Q)$	dataset of a query
$\neg D$	negation of a dataset
$dependsOn(v_1, v_2)$	view dependency
$DISet(I)$	The set of datasets of the views in I

Table 1: Data Structures and Auxiliary Functions

A.1 Datastructures

We start by defining necessary data structures. The semantics is defined on *well founded graphs*, which consist of *positive* and *negative* statements. Note that we only need this construct to define the semantics. We do not need to express it in RDF. Definitions 7 and 8 adjust definitions from [4]. Definitions 11 and 12 extend them to sets of NGs.

Definition 7 (Well Founded Graph).
A polarized statement *is a quadruple of the form:* $R \times U \times R \times \{+, -\}$

A well founded graph G^W *is a set of polarized statements.*

We say the positive subgraph $pos(G^W)$ *of* G^W *is the set* $\{(s,p,o) \mid (s,p,o,+) \in G^W\}$.

We say the negative subgraph $neg(G^W)$ *of* G^W *is the set* $\{(s,p,o) \mid (s,p,o,-) \in G^W\}$.

$pos(G^W)$ *and* $neg(G^W)$ *are disjunct.*

The RDF interpretation of a well founded graph G^W *is* $pos(G^W)$. *The well founded graph of an RDF graph* G *is the set* $\{(s,p,o,+) \mid (s,p,o) \in G\}$.

Definition 7 extends to named graphs and Networked Graphs.

We determine dependencies of views based on the datasets of SPARQL queries.

Definition 8 (Dataset).
A dataset *is a set* $\{G, (n_1, G_1), (n_2, G_2), ...(n_n, G_n)\}$ *where* G *and each* G_i *are well founded graphs, and each* n_i *is a URI. All* n_i *are distinct.*

G *is called the default graph of the dataset, defined using FROM.* (n_i, G_1) *are called named graphs, defined using FROM NAMED. Let* Q *be a SPARQL query, then* $dSet(Q)$ *is the dataset defined in* Q

We will use *negation* and *union* of datasets.

Definition 9 (Negation of a Dataset).
The negation of a dataset D*, written* $\neg D$*, is the dataset for which the following holds:*

If $(n, G) \in D$ *(or* $G \in D$*) then* $(n, G') \in \neg D$ *(or* $G' \in \neg D$ *respectively) such that for all* s, p, o :

Iff $(s,p,o,+) \in G$ *then* $(s,p,o,-) \in G'$ *and iff* $(s,p,o,-) \in G$ *then* $(s,p,o,+) \in G'$.

Definition 10 (Union of Datasets).
The union of datasets D_1 *and* D_2*, written* $D_1 \cup D_2$ *is the dataset for which the following holds: Let* $(n, G) \in D_1$

- *The default graph of* $D_1 \cup D_2$ *is the union of the default graphs of* D_1 *and* D_2.
- *If there is no* $(n, G') \in D_2$*, then* $(n, G) \in D_1 \cup D_2$ *else* $(n, G \cup G') \in D_1 \cup D_2$.

To evaluate a NG G^W we must take into account all graphs, which G^W depends upon.

Definition 11 (Interdependence Set).
We say a view v_1 depends on *another view* v_2*, written dependsOn(*v_1*,* v_2*), if* G *is a graph in the dataset of* v_1 *and* v_2 *is used in a view definition in* G.

An interdependence set I *is a set of NGs, such that for all view definitions* v_1 *and* v_2 *the following holds: If* v_1 *is a view definition in graph* G *and* $G \in I$ *and dependsOn(*v_1*,* v_2*), then all* $G_i \in \mathrm{dSet}(v_2)$ *are also in* I.

Figure 1 in section 4 illustrates the interdependence set containing mikesProject. We refer to all datasets involved in evaluating the semantics of an interdependence set as the *dataset of an interdependence set.*

Definition 12 (DISet).
The dataset of an interdependence set I*, written* $\mathrm{DISet}(I)$ *is the set of the datasets of the views defining graphs in* I*. Graphs with equal names in any two datasets in DISet(*I*) are equal.*

The dataset of the interdependence set shown in figure 1 is {{(`:bobFOAF`, bobFOAF), (`:chrisFOAF`, chrisFOAF)}, {(`:mikesProject`, mikesProject), (`:DBLP`, DBLP)}, {mikesProject}}

A.2 SPARQL Evaluation with Symmetric Negation

We need a small change to the SPARQL semantics in order to treat negation symmetrically, by explicitly negating statements instead of using negation by failure. The following definitions 13 - 15 are taken from [4] and extended to well founded graphs.

The SPARQL specification [4] defines a function *eval(D, GP)* as the evaluation of a graph pattern *GP* with respect to a dataset *D*. Basically *GP* is the graph pattern of a view and *D* is the dataset defined in the view. The range of *eval* is a *solution sequence*

Definition 13 (Solution Mapping/Sequence).
A solution mapping, μ*, is a partial function* $\mu : V \rightarrow T$ *mapping from a set of variables to a set of RDF terms.*

A solution sequence is a list of solution mappings.

Let Q *be a SPARQL query and* μ *a solution mapping.* Q/μ *is the query obtained by replacing every variable* $?x$ *in* Q *by* $\mu(?x)$*, if* $\mu(?x)$ *is defined. We say* μ *is complete for* Q*, if it is defined for all variables of* Q*. Analogously, we define* GP/μ *as the RDF graph obtained by substituting every variable* $?x$ *in a graph pattern* GP *by* $\mu(?x)$.

The solution of a CONSTRUCT query is the set of statements obtained by instantiating the CONSTRUCT pattern using the solution mappings computed by *eval*. *eval* is defined for every operator in the SPARQL algebra. It is defined by recursively applying it to its parameters and using join and select operations on the results. We do not discuss in detail how *eval* is defined, but simply use it as the operator for the evaluation of a single view in a single iteration in the following. However, we need to slightly change the definition of *basic graph pattern matching* in order to use it with well founded graphs. The operator which matches statement patterns against the dataset is *eval(G, BGP)*, where *BGP* is a basic graph pattern consisting only of joins of statement patterns and G is a single RDF graph. In contrast to the other operators, basic graph pattern matching is directly evaluated without subsequent calls of *eval* — here the actual matching against the dataset takes place.

Definition 14 (Statement Pattern).
A statement pattern *is a triple of the form* $(R \cup V) \times (U \cup V) \times (R \cup V)$.

We say a statement pattern S enclosed in an OPTIONAL pattern O of a SPARQL graph pattern P in a query Q is matched negatively *in Q, if the following hold:*

- *?x is a variable not used in statement patterns outside O and at least one of the subject, predicate or object is ?x or S is enclosed in a graph pattern* `GRAPH ?x {... S ...}` *and*
- *P contains a FILTER expression outside O, which contains the filter expression* `BOUND(?x)` *nested in an odd number of negations.*

We say that a statement pattern is matched positively*, if it is not matched negatively.*

The notion of a negatively matched statement pattern formalizes *bound negation*, introduced in section 3. In example 3, `(?x foaf:currentProject :SemWebProject)` is matched negatively, all other statement patterns are matched positively. We redefine basic graph pattern matching, such that negative statements are matched instead of using negation as failure, hence treating negation symmetrically. Negative statements are indirectly introduced in NGs evaluation by negating the largest unfounded set, as we will see below.

Definition 15 (Basic Graph Pattern Matching).
A Basic Graph Pattern is a set of statement patterns.

Let BGP be a basic graph pattern in a query Q and G^W be a well founded graph. Let posBGP *(*negBGP*) be the basic graph pattern containing exactly the positively (negatively) matched statement patterns in BGP.*

$eval(G^W, BGP)$ is a solution sequence. μ is a solution in $eval(G^W, BGP)$, if it is complete for BGP and

- $posBGP/\mu \subseteq pos(G^W)$ *and*
- $negBGP/\mu \nsubseteq neg(G^W)$ *and*
- *if a statement pattern S in BGP matches some statement in $neg(G^W)$ then $S/\mu \in negBGP$, else μ maps at least one variable in S to some skolem value*

This definition of basic graph pattern matching treats positively matched statement patterns as in [4]. The result is empty, if negatively matched statements are known to be negative. For single SPARQL queries, the negative statements are exactly the complement of the statements listed in the RDF graphs in the dataset, so the behavior is also the same here. If, however, a negatively matched statement pattern does *not* match a negative statement (the matched statement is `unknown`), a result with skolem values is obtained. This result then is filtered out, because of the bound negation used in the query. Using standard negation as failure here as in [4], would let the bound negation succeed.

A.3 Well Founded Semantics of NGs

The extended SPARQL evaluation function defined above is used to define the direct consequence operator of the well founded semantics for true statements. For negative statements the direct consequence is defined using the negation of the *greatest unfounded set.*

Definition 16 (Unfounded Set).
Let I be an interdependence set.

We say dataset U is an unfounded set of I, if for every polarized statement S in any graph G^W in U the following hold:

$S = (s, p, o, +)$ is positive and for every view v in G^W and for every complete solution mapping μ for v, such that (s, p, o) is in the CONSTRUCT pattern of v/μ, one of the following holds:

1. *some positively matched statement in the graph pattern of v/μ is negative in I or*
2. *some filter expression in the graph pattern of v, which is not surrounded by an optional pattern, evaluates to false or*
3. *some negatively matched statement in the graph pattern of v/μ is positive in I and no filter expression* ***inside*** *the OPTIONAL surrounding the statement evaluates to false or*
4. *some positively matched statement in v occurs in U.*

The greatest unfounded set of I is the union of all unfounded sets of I.

The second and third case make sure we treat solutions correctly, which in principle meet the unfoundedness conditions but are filtered out by some filter condition, which is not used to define bound negation. We define the direct consequence operator W_I of the well founded semantics for NGs based on unfounded sets and the evaluation function *eval*. W_I assumes all statements in the greatest unfounded set to be negative and thus introduces negative statements to NGs evaluation.

Definition 17 (Transformations T_I, U_I and W_I).
We define three transformations T_I, U_I and W_I mapping from DISets to DISets. Let I be an interdependence set containing a NG G^W and $D = DISet(I)$.

- *$(s, p, o, +) \in G^W$ in all datasets in $T_I(D)$, iff there is a view v defining G^W, such that the result of v evaluated against dset(v) in D using the SPARQL evaluation function* eval *contains (s, p, o).*
- *$U_I(D)$ is the greatest unfounded set of I wrt. D.*
- $W_I(D) = T_I(D) \cup \neg U_I(D)$

Theorem 4. *T_I, U_I and W_I are monotonic.*

Proof: directly from the definitions.

From theorem 4 follows, that T_I has a fixpoint [20].

Definition 18.
The semantics of an interdependence set I is defined as the least fixpoint of $W_I(I)$. The semantics of an RDF graph G in an interdependence set I, deref(G)*, is the set of positive statements assigned to the corresponding well founded graph in the datasets in the DISet of I.*

For the finite case, the least fixpoint is reached in finitely many steps [20]. As NG definitions are finite, the only source of non finiteness can be blank nodes created in the CONSTRUCT patterns of view definitions. All other generated statements can only be made of elements of the finite set of RDF resources used in statements in the DISet. Hence, finiteness of NG evaluation can be checked using a simple syntactic test.