Griwes: Generic Model and Preliminary Specifications for a Graph-Based Knowledge Representation Toolkit

Jean-François Baget, Olivier Corby, Rose Dieng-Kuntz, Catherine Faron Zucker, Fabien Gandon, Alain Giboin, Alain Gutierrez, Michel Leclère, Marie-Laure Mugnier, Rallou Thomopoulos

HAL Id: lirmm-00389753

https://hal.inria.fr/lirmm-00389753v2

Submitted on 22 May 2015

Griwes: Generic Model and Preliminary Specifications for a Graph-Based Knowledge Representation Toolkit

Jean-François Baget[1,2], Olivier Corby[1], Rose Dieng-Kuntz[1], Catherine Faron-Zucker[3], Fabien Gandon[1], Alain Giboin[1], Alain Gutierrez[1], Michel Leclère[2], Marie-Laure Mugnier[2], Rallou Thomopoulos[2,4]

[1] Edelweiss, INRIA Sophia Antipolis Méditerranée,
[2] RCR, LIRMM, UMII, CNRS [3] KEWI, I3S, UNSA, CNRS
[4] IATE Joint Research Unit, INRA Montpellier
[1]{First.Last}@inria.fr, [2]{First.Last}@lirmm.fr, [3]{First.Last}@unice.fr

Abstract. Griwes is an initiative to develop a common model and an open-source freeware platform shared by different graph-based frameworks. We provide an overview of its objectives, architecture and specifications. We detail some of the basic mathematical structures that are used to characterize the primitives for graph-based knowledge representation. We then propose to factorize recurrent knowledge representation primitives that can be shared across specific graph-based languages and we provide a proof of concept by showing how two languages (Simple Conceptual Graphs and RDF) can be described in this framework.

Keywords: graph-based languages, semantic web, platform.

1 Introduction

Graph-based knowledge representation formalisms are more and more common, from Conceptual graphs (CG) [19] which are historical descendants of semantic networks, to more recently proposed representations such as RDF[1], SKOS[1] or Topic Maps[2].

The web is playing an important role in the emergence of these new formalisms and in recent web architectures the RDF graph model became a core layer of the stack of standards[3]. Many knowledge representation frameworks are now used online (RDF, RDFS, SKOS, OWL, GRDDL, RDFa, µFormats, etc.)[1] allowing human and artificial agents to weave graphs describing web resources or just any entity and the relations existing between them. In a recent post[4] Tim Berners-Lee insisted on the graph nature (Giant Global Graph) of the semantic web and the importance of this structure in developing and exploiting the semantic web (*i.e.* the web of data).

[1] W3C Semantic Web Activity http://www.w3.org/2001/sw/
[2] http://www.topicmaps.org/
[3] One Web http://www.w3.org/Consortium/technology
[4] Giant Global Graph, Tim Berners-Lee, http://dig.csail.mit.edu/breadcrumbs/node/215

Reasonings on these different graph formalisms are often very similar. We could share many operations and their implementation across frameworks and even within them on different levels of their models e.g. transitive closure in RDFS class hierarchy, in SKOS concept narrower / broader links, in instances of OWL transitive properties, in CG the concept type hierarchy, etc. In fact when we compare these different languages we can find many similarities. Consider for instance the similarities between RDF/S and CG as underlined in [5] and [1]:

- both models consider assertions as positive, conjunctive and existential;
- both models represent assertions as labeled graphs;
- the class hierarchy (resp. property hierarchy) of RDFS is equivalent to the concept type (resp. relation type) hierarchy of CG;
- properties in RDF/S are first class citizens, declared outside classes just like relations are first class citizens in CG;
- subsumption in RDF/S is equivalent to projection in CG;

The reasonings on these different graph-based frameworks are sometimes also shared with other non graph-based formalisms e.g., databases. [19]

Tools designed and developed for these different graph-based frameworks are tailored to specific languages and/or scenarios and this criticism includes the tools we have been working on in the past years such as Cogitant [9] or Corese [5]. These experiences convinced us that it would be interesting to share these efforts and avoid re-designing and re-implementing the same structures and operators again and again. For this reason we started the project Griwes that stands for Graph-based Representations and Inferences for Web Semantics. The main objective of this initiative is to bootstrap an open-source platform, to share efforts on developing graph-based data structures and algorithms with anyone who wants to contribute. This also implies a proper definition of the considered graph structures shared by the different graph-based formalisms.

In the rest of this article we give an overview of the objectives and architecture of Griwes and we position it w.r.t. other contributions in the field (section 2). We then give some details of the basic mathematical structures that are used to characterize the primitives for graph-based knowledge representation (section 3). We proceed with the layer factorizing recurrent knowledge representation primitives that can be shared across specific graph-based languages (section 4). Finally we provide a proof of concept by showing how two languages (Simple Conceptual Graphs and RDF) can be described in this framework (section 5). We conclude with a discussion on several difficulties and perspectives we identified. These sections are extracted from the working draft of a more detailed research report from Griwes available online[5].

2 Griwes Initiative

This section is an introduction to the Griwes initiative to develop a common model and an open-source freeware platform shared by different graph-based frameworks.

[5] http://www-sop.inria.fr/acacia/project/griwes/

2.1 Objectives of the Griwes initiative

In order to develop a common model and an open-source freeware platform shared by different graph-based frameworks, the objectives of the Griwes initiative can be divided into four kinds of tasks:

- Identification of users' and developers' profiles in the graph-based knowledge modeling communities and semantic web communities, and definition of usage scenarios for the platform;
- Definition of a common representation model shared by different graph-based formalisms and of architectural principles for the organization of the toolkit, allowing the platform to federate contributions and extensions and fostering reuse across graph-based representation models;
- Implementation of the API, interfaces and components in an open-source freeware platform.
- Bootstrapping a community of contributors for this platform (users and developers).

2.2 Architecture of the Griwes toolkit

As summarized in figure 1, the current vision of the framework distinguishes three layers of abstraction and one transversal component for interaction:

- *Structure layer*: this layer gathers and defines the basic mathematical structures (e.g. oriented acyclic labeled graph) that are used to characterize the primitives for knowledge representation (e.g. type hierarchy)
- *Knowledge layer*: this layer factorizes recurrent knowledge representation primitives (e.g. a rule) that can be shared across specific knowledge representation languages (e.g. RDF/S, Conceptual Graphs).
- *Language & Strategy*: this layer is two-sided. One side gathers definitions specific to languages (e.g. RDF triple). The other side identifies the strategies that can be applied to these languages (e.g. validation of a knowledge base, completion of a fact by rules).

The *interaction and interfaces* aspect was deemed transversal to these layers. It gathers events (e.g. additional knowledge needed) and reporting capabilities (e.g. validity warning) needed to synchronize conceptual representations and interface representations. In Griwes, we intend to analyze the requirements of that aspect for each layer as soon as the first draft of these layers is stable.

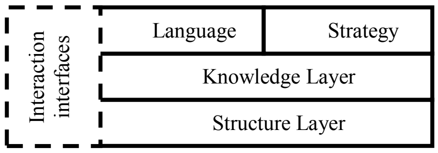

Fig. 1. The three abstraction layers of the current architecture of Griwes.

Before delving into some extracts of the Knowledge and Structure layers, the next section reviews a number of contributions that prefigured, inspired and justified this initiative.

2.3 Related work

There exist a growing number of platforms to reason on graph-based knowledge formalisms, be they in the conceptual graph families or in the RDF graph family.

On2Brocker [7] is an early ontology-based system to handle RDF annotations. Ontologies, queries and rules are expressed in Frame Logic. The query engine translates Frame Logic data into Horn Logic to answer a query. Triple [17] is a query language initially designed for RDF/S and DAML+OIL. Its core is an RDF query language based on Horn Logic extended with syntactical features supporting namespaces, resources and statements (triples). This core language is compiled into Horn Logic programs executed by a Prolog engine. The core Triple language is extended with rules for axiomatizing the semantics of RDFS; they can be used together with a Horn Logic based inference engine to derive additional knowledge from an RDF Schema specification. DAML+OIL or OWL DL cannot be mapped to Horn Logic directly and therefore Triple accesses a Description Logic classifier to handle these extensions. Triple has a layered architecture to handle different knowledge models. Both On2Brocker and Triple remain focused on logic-based engines not exploiting the graph structures of the RDF model.

Sesame [3] is a generic architecture for persistent storing of RDF(S) data into Data Based Management Systems (DBMS) and querying of RDF(S) data with the RQL language. RQL [15] is an RDF query language defined by means of a set of core queries, a set of basic filters and a way to build new queries through functional composition and iterators. When parsing an RQL query, Sesame builds an optimized query tree model from this composition which is then evaluated through a set of calls to the storage and inference modules of Sesame. Sesame supports querying at the semantic level but does not support XML Schema Datatypes, nor does it support inference rules.

DAMLJessKB [13], its successor OWLJessKB and the e-Wallet [8] are tools for reasoning with the Semantic Web and DAML or OWL-Lite. They map the RDF triples and the ontologies into facts of the CLIPS-like language of Jess[6] and apply rules implementing the semantics of RDF, RDFS, XSD and DAML or OWL-Lite. These systems can perform class instance reasoning and terminological reasoning about the relationships among classes. In addition, the e-Wallet is able to run rules to complete the knowledge base, to invoke external services to obtain new knowledge, to answer queries and to control the precision and truthfulness of answers to preserve privacy. Here again these engines remain focused on production rule reasoning not exploiting the graph structure of the RDF model and relying on their internal logic language for query expression.

Jena [4] is one of the most complete platforms offering persistence and reasoning for RDF as well as SPARQL querying. It includes a forward-chaining engine (RETE) and a backward-chaining engine to allow hybrid reasoning and to implement the semantics of RDFS and OWL. Jena relies on a fixed database structure for large storage and on a custom data structure for main-memory storage.

WebKB [14] is an early ontology server and Web robot based on Conceptual Graphs. WebKB interprets and automatically translates into CGs chunks of

[6] JESS engine http://herzberg.ca.sandia.gov/

knowledge statements expressed in a CG linear notation and embedded in Web documents. It also provides commands to query lexical or structural properties of HTML documents or to display specializations or generalizations of a concept or a relation or a CG. OntoSeek [10] is another early system that relies on Conceptual Graphs for ontology-driven content matching. Queries and resource annotations are lexical conceptual graphs to match one against the other. Neither WebKB nor OntoSeek handle RDF(S) data or rules. Moreover they both focus on a specific family of web applications not aiming at allowing different mapping to their graph-based representations and not providing a generic expressive query language.

With the OWL recommendation at W3C, Description Logics (DL) became especially important in the spectrum of logic-based systems on the web. Several systems exist here: Fact and its successor Fact++ [20], KAON2 [16], KAON that remains focused on RDFS, Racer [11] and Pellet [18]. These engines offer classical DL operations such as identification, classification and validation. Queries are usually limited to conjunctive queries and the graph structure of the RDF model is not exploited at the core of these engines.

To summarize, none of these contributions is offering a pivot model and an open-source platform to efficiently implement querying and reasoning on graph-based models. Most of them are tied to specific languages, logics or even applications.

Members of Griwes also developed platforms of their own over the last decade. Let us mention two of them: Cogitant [9] dedicated to conceptual graph reasoning and Corese [5] dedicated to a conceptual graph operationalisation of RDF/S.

Our own tools based upon CGs implementations and also contributions like Amine [12] relying on a combination of Prolog and CGs, suffer from their closed design preventing reuse and cross-pollination. The next section is a guided tour of some extracts of the specifications of Griwes as defined in the current working draft of its research report.

3 Structure layer

The structure layer is the core layer of the architecture of Griwes. We extracted here some definitions of the basic mathematical structures that we chose to characterize the primitives for knowledge representation.

3.1 ERGraphs: Entity-Relation graphs.

Our core representation primitive is intended to describe a set of entities and relationships between these entities; it is called an Entity-Relation graph (ERGraph in short). An entity is anything that can be the topic of a conceptual representation. A relationship, or simply relation, might represent a property of an entity or might relate two or more entities.

The relations can have any number of arguments including zero and these arguments are totally ordered. In graph theoretical terms, an ERGraph is an *oriented hypergraph*, where nodes represent the entities and hyperarcs represent the relations on these entities. However, a hypergraph has a natural graph representation associated with it: a bipartite graph, with two kinds of nodes respectively representing entities and relations, and edges linking a relation node to the entity nodes arguments of the relation; the edges incident to a relation node are totally ordered according to the order on the arguments in the relation.

The nodes (Entities) and hyperarcs (Relations) in an ERGraph have labels. At the structure level, they are just elements of a set L that can be defined in intension or in extension. Labels obtain a meaning at the knowledge level.

Definition of an ERGraph: An ERGraph relative to a set of labels L is a 4-tuple $G=(E_G, R_G, n_G, l_G)$ where

- E_G and R_G are two disjoint finite sets respectively, of nodes called entities and of hyperarcs called relations.
- $n_G : R_G \rightarrow E_G^*$ associates to each relation a finite tuple of entities called the arguments of the relation. If $n_G(r)=(e_1,\dots,e_k)$ we note $n_G^i(r)=e_i$ the ith argument of r.
- $l_G : E_G \cup R_G \rightarrow L$ is a labelling function of entities and relations.

In some knowledge representation primitives and some algorithms it is useful to distinguish some entities of a graph. For this purpose we define a second core primitive, called λ–ERGraph.

Definition of a λ-ERGraph: A λ-ERGraph λ_G is a couple of an ERGraph G and a tuple of entities of G: $\lambda_G = ((e_1,\dots e_k), G), e_i \in E_G$. We say that k is the size of λ_G and that $(e_1,\dots e_k)$ are distinguished in G.

Definition of an induced SubERGraph: Let $G=(E_G, R_G, n_G, l_G)$ be an ERGraph. Let $E_{G'}$ be a subset of E_G. The SubERGraph of G induced by $E_{G'}$ is the ERGraph $G'=(E_{G'}, R_{G'}, n_{G'}, l_{G'})$ defined by: (1) $R_{G'}= \{ r \in R_G \mid \forall 1 \leq i \leq card(n_G(r)), n_G^i(r) \in E_{G'} \}$ (2) $n_{G'}$ is the restriction of n_G to $R_{G'}$ (3) $l_{G'}$ is the restriction of l_G to $E_{G'} \cup R_{G'}$

Definition of a Merge: let $G=((g_1,\dots g_k), G')$ et $H=((h_1,\dots h_k), H')$ two λ-ERGraphs of same size, the merge of H in G modifies G' by adding a copy $C(H')$ of H' to G' and then for $1 \leq i \leq k$ by merging the entities $C(h_i)$ and g_i.

Note that the labels of the merged entities are obtained by applying a method defined at higher levels.

3.2 Mapping between ERGraphs

Intuitively, a Mapping associates entities of a query ERGraph to entities of an ERGraph in a knowledge base of ERGraphs. Mapping entities of graphs is a fundamental operation for comparing and reasoning with ERGraphs.

Definition of an EMapping: Let G and H be two ERGraphs, an EMapping from H to G is a partial function M from E_H to E_G *i.e.* a binary relation that associates each element of E_H with at most one element of E_G ; not every element of E_H has to be associated with an element of E_G.

Definition of an ERMapping: Let G and H be two ERGraphs, an ERMapping from H to G is an EMapping M from H to G such that: Let H' be the SubERGraph of H induced by $M^{-1}(E_G)$, $\forall r' \in R_{H'}$ $\exists r \in R_G$ such that $card(n_{H'}(r')) = card(n_G(r))$ and $\forall 1 \leq i \leq card(n_G(r))$, $M(n_{H'}{}^{i}(r')) = n_G{}^{i}(r)$. We call r a support of r' in M and note $r \in M(r')$

Mapping is a basic operation used in many more complex operations e.g. rule application. Let us note that by default an EMapping is partial. This enables us to manipulate and reason on EMappings *during* the process of mapping graphs. When this process is finished, the EMapping – if any – is said total: all the entities of the query graph H are mapped. In general we use specific mappings that preserve some chosen characteristics of the graphs (e.g., compatibility of labels, structural information etc.); figure 2 shows their hierarchy.

In particular an ERMapping constrains the structure of the graphs being mapped and an EMapping$_{<X>}$ constrains the labelling of entities in the graphs being mapped. An **ERMapping** is an EMapping that leads to map each relation in H to a relation in G with the same arity. An **EMapping$_{<X>}$** is an EMapping that satisfies a compatibility relation X on entities labels. An **ERMapping$_{<X>}$** is both an ERMapping and an EMapping$_{<X>}$. A Homomorphism is a total ERMapping. Other specializations include: injective mappings, surjective mappings, faithful mappings (preserve the absence of hyperarcs), etc.

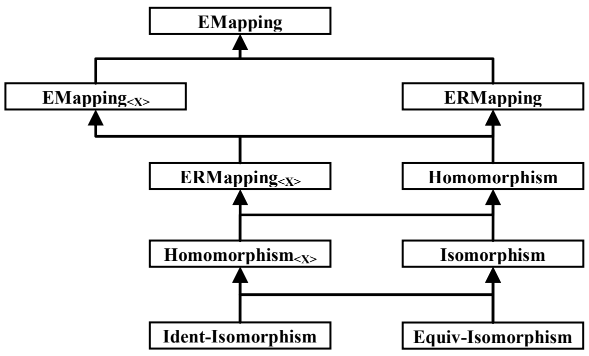

Fig. 2. EMapping specialization hierarchy.

In conceptual graph projections, many systems map not only entities, but relations as well. The notion of projection as defined in conceptual graphs corresponds to a Homomorphism$_{<X>}$ that is to say a total ERMapping$_{<X>}$, where X is a preorder over the label set L.

3.3 Proofs of a Mapping

We define the proof of a mapping as a kind of "reification" of the mapping; a proof provides a static view over the dynamic operation of mapping, enabling thus to access information relative to the *state* of the mapping. Formally the proof of a mapping is the set(s) of associations detailing the exact association from each entity and relation of the query graph H to entities and relations of G.

We follow the hierarchy of mappings outlined in the previous section and associate with each kind of EMapping a notion of proof: EProof, ERProof and ERProof$_{<X>}$. For instance the proof for a homomorphism corresponds to the proof of a total ERMapping$_{<X>}$ where X is a preorder over the label set L and defined as follows:

Definition of an EProof: Let G and H be two ERGraphs, and M an EMapping from H to G. The EProof of M is a set $M_E = \{ (e_H,e_G) \in E_H \times E_G \mid e_G=M(e_H) \}$.

Definition of an ERProof: Let G and H be two ERGraphs, and M an ERMapping from H to G. Let H' be the SubERGraph of H induced by $M^{1}(E_G)$. An ERProof of M is a couple $P=(M_E,M_R)$ where M_E is the EProof of M and $M_R= \{(r_1,r'_1),\dots (r_k,r'_k)\}$ with $\{r_1,\dots,r_k\}=R_{H'}$ and $\forall 1 \le i \le k \; r'_i \in M(r_i)$.

Definition of an ERProof$_{<X>}$: Let G and H be two ERGraphs, and M an EMapping from H to G. An ERProof$_{<X>}$ of M is a couple $P=(M_{EX},M_{RX})$ where M_{EX} is the EProof$_{<X>}$ of M and $M_{RX}= \{(r_1,r'_1,p_1)\dots (r_k,r'_k,p_K)\}$ where $\{(r_1,r'_1)\dots (r_k,r'_k,)\}$ is the second element of an ERProof of M and $\forall 1 \le i \le k \; p_i$ is a proof of $(l_G(M(r)), l_H(r)) \in X$.

At this point we make no assumption on the structure of p_i and the means to obtain it. A system for comparing labels should be able to produce such proofs, e.g. a chain of subsumption relations which transitive closure confirms the comparison of two labels. Note that several different ERProofs can be associated to a same ERMapping (e.g. when there are two twin relations in G that can support a same relation of H).

3.4 Constraints System for Mappings

An EMapping constraint system is a function $\mathcal{C}$ that sets additional conditions that an EMapping must satisfy in order to be correct.It takes the form of an evaluable expression which must evaluate to true for an EMapping to satisfy the constraint system.

Definition of an EMapping Constraint System: An EMapping constraint system for an EMapping M from H to G is a function $\mathcal{C}(E)$ where E is the triple (H,P,V) called the environment, with P the proof of M and V a binary relation associating to variables v_i a unique entity or relation of H. This function can evaluate to *{true, false, unknown, error}*.

An EMapping M satisfies (resp. violates) a constraint system $\mathcal{C}$ if $\mathcal{C}(M)=true$ (resp. if $\mathcal{C}(M)=false$).

This facet of the specifications was motivated by scenarios using expressive query languages such as SPARQL [6]. For instance, let us consider the following SPARQL query and in particular its FILTER clause (line 7):

```
1. PREFIX inria: <http://www.inria.fr#>
2. SELECT ?student ?name
3. WHERE {
4.  ?student rdf:type inria:Student
5.  ?student inria:name ?name .
6.  ?student inria:age ?age .
7.  FILTER (xsd:integer(?age) > 22 && regex(?name, "A.*")) }
```

The triples of the query pattern can be seen as a graph pattern requesting students (line 4) with their name (line 5) and their age (line 6):

```
[Student]-
   (name)->[?name]
   (age)->[?age].
```

Line 7 however is an additional constraint pattern that has to be satisfied in order for the matching to be correct; it specifies that the integer value of the age has to be greater than 22 and that the name should start with an "A".

These kinds of constraints motivated the definition of constraint systems in our specifications but constraint systems are also envisaged to provide efficient access means to indexes of graphs, for instance to retrieve all the arcs of a graph satisfying a given constraint system.

4 Knowledge Layer

In our architecture, a knowledge base B is defined by a vocabulary, one or several bases of facts, optionally a base of rules and a base of queries. *B= (Vocabulary, Fact Base* $^{+}$*, Rule Base*, Query Base*).*

A vocabulary is a set of none necessarily disjoint named sets of elements called vocabulary subsets together with preorders on the union of these sets:

Definition of a Vocabulary: A Vocabulary V is a tuple $V = \left(U = \bigcup_{1 \le i \le k} V_i, \le_1, \ldots, \le_q \right)$ where V_i are k sets of elements and $\le_i$ are q preorders on U.

Definition of a Fact: A Fact is an ERGraph.

Definition of a Base of Facts: A Base of Facts is a set of Facts.

Let us note that every ERGraph G in a base of facts respects $l_G : E_G \cup R_G \rightarrow L$ where L is constructed from the set U of elements of the vocabulary V of the knowledge base.

Definition of a Query: A Query is a couple $Q=(q, \mathcal{C})$ of a λ-ERGraph $q=((e_1,\ldots e_k), G)$ and a Constraint system $\mathcal{C}$.

The answers to a query depend on the kind of EMapping used to query the base. In the next definitions, the letter X stands for a type of EMapping;

X-Answer to a Query: Let $Q=(((e_1,\ldots e_k), G), \mathcal{C})$ be a query and F be a Fact. $A=(a_1,\ldots a_k)$ is an X-Answer to Q in F iff there exists an EMapping M of type X from G to F satisfying $\mathcal{C}$ such that $M(e_i)=a_i$.

Note: the proof of an X-Answer is the proof of the EMapping associated to that X-Answer.

Definition of a Base of Queries: A Base of Queries is a set of Queries.

Definition of a Rule: A Rule is a couple *R=(H,C)* of a Query $H=(G, \mathcal{C})$ and a λ-ERGraph *C* of the same size as *G*. *H* is the hypothesis of the rule, and *C* is its conclusion.

***X*-applicable Rule:** A rule *R=(H,C)* is *X*-applicable to a fact F iff there exists an X-Answer to H in F.

X-applying a Rule: Let *R=(H,C)* be a rule X-applicable to a fact *F*, and *A* be an X-Answer to *H* in *F*. The X-Application of *R* on *F* with respect to *A* merges *C* in *(A,F)*.

Definition of a Base of Rules: A Base of Rules is a set of Rules.

Definition of an ERFunction: An ERFunction *F* is a function associating to an ERProof *P* a label or an error.

Definition of a functional ERGraph: A functional ERGraph is an ERGraph where some entities or relations are labelled with ERFunctions.

Evaluation of a functional ERGraph: The evaluation of a functional ERGraph *G* with respect to an EProof *P* and an environment *E* is a copy *G'* of *G* where every functional label is replaced by the evaluation of the function against P. If any of the evaluations returns an error then $G'=\varnothing$.

Definition of a Functional Rule: A functional rule is a rule *R=(H,C)* where *C* is a functional λ-ERGraph.

X-applying a Functional Rule: let *R=(H,C)* be a functional rule X-applicable to a fact *F*, and *A* be an X-Answer to *H* in *F* and P be a proof of that X-Answer. The X-functional-Application of *R* on *F* with respect to *P* merges the evaluation of *C* with respect to *P* in *(A,F)*.

Definition of Co-Reference: A Co-Reference relation *R* is an equivalence relation over the set of entities of *G*..

Definition of a Normal Form: let *G* be an ERGraph with a co-reference relation *R* and a function $fusion(E_1,E_2,\ldots, E_n)$ that returns a new entity from a set of entities, the normal form of *G* is the graph *NF(G)* obtained by merging every entities of a same equivalence class defined by *R* as a new entity calculated by calling *fusion* on the entities of this class.

Co-reference and fusion are abstract functions which must be specified at the language level.

5 Validating Against two Languages: Simple Graphs and RDF

This article focuses on the structure layer and the knowledge layer of Griwes and does not include a description of the language and strategy layer still under discussions. However this section shows how the primitives of the pivot model defined in Griwes can be used to represent the semantics of two languages: Simple (conceptual) Graphs and RDF. This practice would, ultimately, be the objective of the language layer.

5.1 Representing Simple Graphs in the Griwes model

Non-surprisingly, the SG [2] graphs map smoothly to the core model of Griwes since this model was inspired by the conceptual graphs formalism.

Primitive SG	Griwes translation
Primitive concept type	Member of a specific finite vocabulary sub-set *TC* defined in extension. This finite vocabulary sub-set has a partial order $\leq_{TC}$.
Primitive relation type	Member of a specific finite vocabulary sub-set *TR* defined in extension and providing a label *l*, an arity *k*, and a signature $s \in (TCC)^k$. This finite vocabulary sub-set has a partial order $\leq_{TR}$ defined only for labels with the same arity.
Conjunctive concept type	Member of a specific vocabulary sub-set *TCC* defined in intension; sub-set of power set of Primitive concept types. This finite vocabulary sub-set has a partial order $\leq_{TCC}$ derived from $\leq_{TC}$. NB: $TC \subset TCC$
Individual marker and Generic marker *	Member of a specific finite vocabulary sub-set $M=I \cup \{*\}$ defined in intension. This finite vocabulary sub-set has a partial order $\leq_M$ such that $\forall i \in M \;\; i \leq_M *$.
Concept	An entity where the label is a couple (t, m) with $t \in TCC$ and $m \in M$. We define $\leq_C$ a partial order on these labels such that $(t_1, m_1) \leq_C (t_2, m_2)$ iff $t_1 \leq_{TCC} t_2$ and $m_1 \leq_M m_2$.
Relation	a relation where the label is a type $t \in TR$
Fact	*A Fact.*
Simple Graph	An ERGraph respecting labelling functions.
Query	A query $Q=(q, C)$ with $\mathcal{C}=\emptyset$.
Rule	A rule R=(H, C) with $\mathcal{C}=\emptyset$.
Banned concept type	Member of a specific vocabulary sub-set *BT* sub set of power set of primitive concept types; members of this sub-set should never be used in other sets of the vocabulary, in facts, in queries or rules.
Support	the vocabulary *V*.
Graph specialization	Let $\leq$ be the partial order defined by $\leq_C$ when applied to two entities, by $\leq_{TR}$ when applied to two relations, and not holding for any other case. A graph *G* specializes a graph *H* if there exists a homomorphism$_{\leq}$ from *H* to *G*.
Graph deduction	*H* is deduced from *G* iff the normal form *NF(G)* specializes *H* or *G* is inconsistent; *NF(G)* is defined by $coref_{SG}$ and $fusion_{SG}$.

5.2 Representing RDF in the Griwes model

This section shows how the RDF graph model can be mapped to the core model of Griwes. Mappings given in the following table rely on the following preorder.

Definition: let $\leq_{RDF}$ be a preorder over *V* such that

- $x \leq_{RDF} y$ if $y \in$ Blanks
- $x \leq_{RDF} y$ if $x, y \in Literals$[2] and value(x)=value(y)
- $x \leq_{RDF} y$ if $x=y$

Primitive RDF	Griwes translation
Blank	Member of a specific vocabulary sub-set defined in intension.
Literal	Member of a specific vocabulary sub-set defined in intension.
Literal ^^datatype	Member of a specific vocabulary sub-set defined in intension.
Literal @lang	Member of a specific vocabulary sub-set defined in intension.
URI ref	Member of a specific vocabulary sub-set defined in intension.
Triple: subject, predicate, object (x p y)	a relation in an ERGraph ; it would naturally be binary but

	additional coding information may be added with n-ary relations e.g. quadratic relations specifying the source and the property reified. The ERGraph *G* includes the relation R_p such that $n_G(R_p)=(e_x,e_p,e_y)$
RDF graph G (i.e. a **set** of triples on a given vocabulary)	An ERGraph such that for each distinct term *t* appearing in a triple of *G* the ERGraph *E* associated to *G* contains a distinct entity *e(t)* and for each triple *(s,p,o)* of *G*, *E* contains a relation *r* such that $n_E(r)=(e(s),e(p),e(o))$. Remark : a well-formed RDF ERGraph: - has no isolated entity; - first element of relations must not be a Literal; - a property name is only a URI ref; One may have to work on non-well-formed RDF ERGraph.
RDF nodes	Entities appearing in position 1 and 3 of a relation.
Vocabulary (set of names)	Vocabulary.
RDF Vocabulary (rdf:Property, rdf:type)	a specific vocabulary sub-set defined in extension for RDF.
Simple RDF entailment	H entails G iff there exists a $\text{Homomorphism}_{\leq \text{RDF}}$ from G to the normal form NF(H) defined by $coref_{RDF}$ and $fusion_{RDF}$.
RDF axioms	the ERGraph representation of the triples of the axiomatic triples of RDF are asserted in every base of facts.
x rdf:type t	as any other triple. (NB: *t* can be integrated in the label of the entity representing *x*)

RULE 1
IF *x p y* in RDF graph *G*
THEN *p rdf:type rdf:Property*
	R=(H,C) where *H=((e(y)),H')* with *H'* is the graph associated with *{(x,y,z)}* where *x*, *y* and *z* are blanks and *C=((e(u)),C')* with *C'* the graph associated with *{(u, rdf:type, rdf:Property)}* where *u* is a blank and *rdf:type* and *rdf:Property* are URI refs of the RDF vocabulary.

RULE 2
IF *x p y^^d* in RDF graph *G*
and *y^^d* well-typed
THEN *y^^d rdf:type d*
	R=(Q,D) a functional rule, where *Q=(H,C)* with *H=((e(z)),H')* with *H'* is the graph associated with *{(x,y,z)}* where *x*, *y* and *z* are blanks, C is satisfied iff e(z) is labelled by a well-typed datatype literal. *D=((e(a)),D')* is the lambda functional ERGraph associated with *{(a, rdf:type, fun:getType(im(e(z)))), (x, fun:id(im(r(y)), fun:getNormalForm(im(e(z)))) , (fun:getNormalForm(im(e(z))), rdf:type, fun:getType(im(e(z)))) }* where *a* is a blank and *rdf:type* is a URI ref of the RDF vocabulary and *fun:getType()* is a function extracting the type from a literal.

6 Discussion

In this article we presented an initiative to design a common model and specify a platform to share state-of-the-art structures and algorithms across several graph-based knowledge representation frameworks such as RDF/S, Conceptual Graphs, Topic Maps or SKOS. This article is extracted from the working draft of a more detailed research report from Griwes available online[7].

We identified a number of limitations and problems that we intend to address in a near future:

[7] http://www-sop.inria.fr/acacia/project/griwes/

- **Generalization of lambdas to relation labels:** we may have to consider two tuples in lambda graphs, a tuple of entities and a tuple of relations (or a tuple of entities and relations) in order to use variables on relations as allowed in SPARQL.
- **Structure of proofs:** at this point of the design, we made no assumptions on the structure of the proofs and the means to obtain them; this may have to be detailed in the future and extended to reasoning in general.
- **Index of graphs:** in order to wrap different efficient accesses to graphs and also heterogeneous arc producers (e.g. database wrappers) we are currently working on introducing indexes as companion structures of graphs that provide constrained listing of the components of a graph to support efficient access mechanisms.
- **Relations with different arities:** in the ERMapping, we may have to generalize the constraint on arity and matching for instance to map relations with different arities or different orders in their arguments.
- **Complex modifiers in queries:** a query language like SPARQL introduces constructors for representing optional parts in queries, disjunctive parts, constraints with complex scopes, constraints between different answers to a query, etc. These extensions will require additional work.
- **Architectural choices:** for instance there is an ongoing discussion on the status of queries and the fact they should or should not be linked to knowledge base.
- **Subtleties in domains of interpretations:** the distinction between terms and values in SPARQL-RDF is full of complex cases that require us to find the right compromise between efficiency and size of data.

To illustrate these questions, let us just detail this last example to consider the options one could have in representing datatyped literals and their value. Currently RULE 2 of the RDF mapping presented here does not cover coreference between a Literal Entity and its datatyped value representation. We identified three solutions to this problem:

- Explicitly indicate coreference between these entities and handle them in the algorithms;
- Consider composite labels representing sets of literals and modify preorders on labels and normalization so as to indicate original destinations of arcs on the arcs themselves;
- Use hyperarcs containing the literal representation, its type and its value and modify ERMappings to handle a variable number of arguments in the arc.

The current work in Griwes includes discussing these options and finding the right compromise between efficiency, generality and feasibility.

To summarize, we now have a first draft of three layers of our architecture. We intend to refine and extend this architecture and, even more importantly, to start the open-source design of the corresponding APIs and their implementations.

Acknowledgments. We are grateful to the COLOR funding program of INRIA.

References

1. Baget, J.B., RDF entailment as a graph homomorphism, Proceedings of the. 4th International Semantic Web Conference, pp 82-96, LNCS, Springer (2005).
2. Baget, J.-F., Mugnier, M.-L., The SG Family: Extensions of Simple Conceptual Graphs. IJCAI, pp. 205-212 (2001)
3. Broekstra, J., Kampman, A., van Harmelen, F., Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In ISWC'2002, pp. 54-68, (2002)
4. Carroll, J. J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., and Wilkinson, K., Jena: Implementing the semantic web recommendations. Technical Report HP Lab (2003).
5. Corby, O., Dieng-Kuntz, R., Faron-Zucker, C.: Querying the Semantic Web with the Corese Search Engine, ECAI, IOS Press pp. 705--709 (2004)
6. Corby, O., Faron-Zucker, C., Implementation of SPARQL Query Language based on Graph Homomorphism, In Proc. 15th International Conference on Conceptual Structures (2007)
7. Fensel, D., Angele, J., Decker, S., Erdmann, M., Schnurr, H.-P., Staab, S., Studer, R., and Witt, A., On2broker: Semantic-based access to information sources at the WWW, in World Conference on the WWW and Internet. (1999)
8. Gandon, F., Sadeh, N., Semantic Web Technologies to Reconcile Privacy and Context Awareness, In Web Semantics: Science, Services and Agents on the World Wide Web, Elsevier Science, vol 1(3), pp. 241--260 (2004)
9. Genest, D., Salvat, E., A Platform Allowing Typed Nested Graphs: How CoGITo Became CoGITaNT, Proc. 6th International Conference on Conceptual Structures, Montpellier, France, August 98. Lecture Notes in AI, volume 1453, pp. 154--161. Springer, (1998)
10. Guarino, N., Masolo, C., Vetere, G., Ontoseek: Content-based access to the Web. In IEEE Intelligent, Systems, vol. 14(3), pp. 70--80 (1999)
11. Haarslev, V., Möller, R., Racer: An OWL Reasoning Agent for the Semantic Web. In Proceedings of the International Workshop on Applications, Products and Services of Web-based Support Systems, Halifax, Canada, October 13, pages 91--95 (2003)
12. Kabbaj A., K. Bouzoubaa, K. ElHachimi and N. Ourdani, Ontology in Amine Platform: Structures and Processes, in the 14th Proc. Int. Conf. Conceptual Structures, ICCS 2006, Aalborg, Denmark. (2006)
13. Kopena, J., Regli, W., DAMLJessKB: A Tool for Reasoning with the Semantic Web, IEEE Intelligent Systems, pages 74-77, May/June, volume 18, number 3 (2003)
14. Martin, P., Eklund, P., Knowledge Retrieval and the World Wide Web. In IEEE Intelligent, Systems, vol 15(3), pp. 18--25 (2000)
15. Miller, L., Seaborne, A., Reggiori, A., Three Implementations of SquishQL, a Simple RDF Query Language. In Proc. of the 1st International Semantic Web Conference, ISWC2002, LNCS 2342, pp. 423-435, Sardinia, Italy, (2002)
16. Motik and Ulrike Sattler, KAON2, A Comparison of Reasoning Techniques for Querying Large Description Logic Aboxes, Boris Book: Logic for Programming, Artificial Intelligence, and Reasoning, LNCS Volume 4246 (2006)
17. Sintek, M., Decker, S., Triple: A Query, Inference and Transformation Language for the Semantic Web. Proc. of the 1st International Semantic Web Conference, ISWC 2002, LNCS 2342, pp. 364--378, Sardinia, Italy (2002)
18. Sirin, E., Parsia, B., Bernardo Cuenca Grau, Aditya Kalyanpur and Yarden Katz. Pellet: A practical OWL-DL reasoner, Journal of Web Semantics, 5(2) (2007).
19. Sowa, J. F. "Conceptual graphs for a database interface," IBM Journal of Research and Development 20:4, pp. 336--357 (1976).
20. Tsarkov, D., Horrocks, I., FaCT++ Description Logic Reasoner: System Description, LNCS Volume 4130, (2006)