Creating or Using various Knowledge Representation Models and Notations

Philippe Martin, Jérémy Bénard
EA2525 LIM, University of La Réunion, F-97490 Sainte Clotilde, France
Philippe.Martin@univ-reunion.fr (also adjunct researcher of Griffith University, Australia)
Jeremy.Benard@logicells.com

Abstract: There are many knowledge representation (KR) languages (KRLs), i.e., many KRL notations and KRL abstract structure models. They suit different needs. E.g., knowledge modeling and sharing require expressive and concise KRLs to support and ease the entering of precise knowledge. Many KRLs are more suited to knowledge exploitation with computational tractability constraints. Current KR-based tools – including KR translators – allow the use of only one or few KRLs, and hardly allow their end-users to *adapt* these KRLs to their needs, e.g., the need to exploit even ad hoc KRs. Indeed, some *systematic* ad hoc usages can be automatically interpreted. Finally, it is difficult to compare KRLs and KRs according to criteria or KRL related best practices. The approach presented in this article addresses these problems by answering an *original* research question: "can KR import or export methods be specified in a *generic* way and, if so, how can they and their resources be specified?". The approach is based on an *ontology of KRLs,* hence on KRs about KRLs. It is here named KRLO. It has three original features: i) it represents very different KRL abstract models in a uniform way, ii) it represents KRL notations, and iii) it specifies methods for importing and exporting KRs, and hence also translating them. This article presents principles and uses for this approach. We have built Javascript functions and tools that import and export KRs by exploiting KRLO and a parser generator. For these tools to use new KRLs or KRL presentations, their end-users can add or adapt specifications in KRLO. Other tools can use these tools or functions as Web services or modules. No translator between each pair of KRLs needs to be written. At least for export purposes, KRLO can *also* be exploited via inference engines for OWL2 or Datalog, or via simple path retrieval mechanisms, e.g., via SPARQL queries.

Keywords: knowledge representation, importing, translation, exporting and sharing

1. Introduction

KRLs are languages enabling to represent information in logic-based forms – *knowledge representations (KRs*) – within knowledge bases. KRs are exploited by inference engines or, more generally, knowledge management (KM) systems, e.g., for precision-oriented knowledge sharing, retrieval or management and for problem solving.

The word "KRL" has three meanings. First, a KRL *abstract model,* e.g., RDF+OWL2 or MOF. Such models are *not* those of model-theoretic semantics. Second, a KRL *notation, i.e., a* concrete model or syntax, e.g., Turtle or MOF-HUTN. Third, a combination of the last two, e.g., RDF/XML (i.e., "RDF linearized with XML" abbreviated using "/" as often done by the W3C – World Wide Web Consortium), RDF+OWL2QL/Turtle and MOF/MOF-HUTN. From now on, the word "model" refers to "KRL abstract model". Many KRLs exist. Each suits some constraints or preferences. E.g., for *modeling and sharing* complex information, experienced *KR designers* are likely to prefer *high-level* KRLs. In this article, "more high-level" means i) "more concise", i.e., requiring "less constructs", and ii) more "normalizing", i.e., better "helping to represent information in automatically comparable ways". Higher-level KRLs are also more expressive. From high-level KRs, lower-level ones can be derived.

Ideally, there would exist KRL translators to let KM tools use any knowledge base and, ideally, these tools would let their users select and adapt any KRL they wish for entering and displaying knowledge. This is still far from being the case. There are few KRL translators and most translate only between few notations associated to one standard model, e.g., the translators based on the OWL API (Horridge and Bechhofer, 2011). Furthermore, these KRLs are not high-level, as illustrated in Section 2 and by Guizzardi et al (2010), and these tools do not translate between some different ways these KRLs are used (Beek et al, 2014) (Färber et al, 2015).

To solve those problems and enable KM tool developers to avoid the lengthy task of implementing import, export or translation features from/to every model and notation, i) we have created KRLO, an *ontology* of KRLs, i.e., we have created KRs representing KRLs and helping to represent or exploit KRLs, and ii) we have developed Javascript functions and tools exploiting this ontology. KRLO has at least two original features: i) it represents – and enables people to represent – very different models in a simple and uniform way, and ii) it includes specifications for notations. KRLO and its associated tools solve the following original research problem: can *generic* "KR import or export methods" be specified and, if so, how can they and their resources be specified? Here, "generic" means "usable for any specified KRL".
Since KRLO is only about KRLs, it can help aligning or merging ontologies only if they use different KRLs.
Because of space restrictions and the target audience – i.e., KM interested people, not KR focused researchers – this article is a *synthesis* article: it describes principles and gives examples. The content of KRLO, its associated Javascript tools and an extension of this article are accessible from http://www.webkb.org/kb/it/o_knowledge/o_KRL/. WebKB-2 (Martin, 2009), a *shared ontology* server, enables Web users to *extend* our version of KRLO and hence *share their extensions of it.* Alternatively, people may copy KRLO onto their computers and *adapt it to their needs*. In the context of a collaboration with us, they can also do so with our tools.
Section 2 introduces KRLO and its associated tools. It also gives examples of KRL translations, thus of KR imports and exports. Section 3 introduces the principles of our approach. Section 4 presents our validation methods, the kinds of use cases for our approach, and comparisons to other works.

2. Examples of content and uses of KRLO

2.1 IKLmE, a model to define complex constructs

To *define* some of its types, KRLO first specifies IKLmE: "IKL model and Extensions based on it". IKL (Hayes, 2006) is a KRL designed for interoperability: it has the classical First Order Logic (FOL) semantics but still permits the use of many constructs of Higher-Order Logics (HOL). E.g., IKL gives a well-defined semantics to meta-statements that are *contexts*, i.e., "statements specifying conditions for their embedded statements to be true".
Using the primitives of IKL, IKLmE also *defines* types helping to represent complex statements. Thus, as Table 2 will illustrate, IKLmE can be used to complement any model for representing KRs that require HOL-like FOL-compatible constructs. E.g., IKLmE defines different kinds of meta-statements and numerical quantifiers. IKLmE answers the following research problem: how to represent common HOL constructs using only a small set of binary relations.
To complement RDF, the W3C proposes the OWL2 ontology which *declares* types for constructs belonging to various description logics (w3.org/TR/owl2-profiles). However, before IKLmE, no model ontology existed for HOL-like constructs.

2.2 Methods in KRLO and associated tools

KRLO specifies a generic method for importing KRs in a given KRL and another one for exporting KRs in a given KRL. These method specifications are logic-based and thus could potentially be exploited by inference engines. Tools doing so could let their end-users dynamically adapt method specifications, not just KRL specifications. However, at least for KR parsing, these tools would likely be very slow. Thus, we have not yet validated this part. Instead, we have *implemented* these methods in Javascript. The resulting functions and tools can be used as Web services or modules in other tools.

2.3 Translation examples

Table 1 illustrates translations on a *simple* statement: its expressiveness is covered by OWL2-QL (w3.org/TR/owl2-profiles). It includes two "links", i.e., "triples or property instances" in RDF terminology (w3.org/RDF), "binary relations" in some other graph-based KRL terminologies (Sowa, 1984) and "binary predicates" in FOL. For readability purposes, link type names begin by "r_" and are in italics, and we use the terminology, normalization conventions and basic link types of Sowa (1984). E.g., the names of the concept types – i.e., in FOL, the types of elements that can be linked by predicates – begin by an uppercase and are nominal expressions. Thus, links of the infixed form "X r_y Z" or of the prefixed form "r_y (X, Z)" can be read "X has for r_y Z", while a link of the form "X r_y of Z" can be read "X is r_y of Z". The FE (Martin, 2002) notation makes this reading explicit. The FL (Martin, 2009) notation enables the writing or presentation of any number of KRs as a connected graph, with links in prefix, infix or postfix forms.
In Table 1, "FOLwDef" refers to "FOL with definitions". With the first four notations, FOLwDef is indicated as model since these notations have the syntactic sugar to support it and since using it leads to higher-level KRs than using link types from OWL2-QL. Four of the notations are mainly prefixed: Peano-Russel, KIF (Knowledge Interchange Format; logic.stanford.edu/kif/kif.html) and OWL_Functional-style (w3.org/TR/owl2-syntax). The others are mainly infixed. Table 1 uses Turtle (w3.org/TR/turtle) since it is close to JSON-LD (json-ld.org) and reused in SPARQL (w3.org/TR/sparql11-update).

Table 1: Representation of *a purposely oversimplified English statement* in different KRLs.

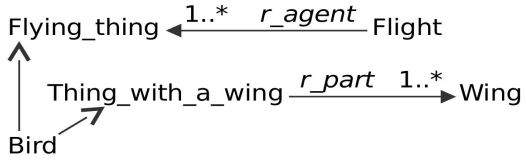
English: By definition (of the term "bird"), any bird flies and has at least a wing.
FOLwDef. / **Peano-Russel** ("/" means "linearized with"): Bird =def ∀ b (Bird(b) => ∃ f,w Flight(f) ∧ Wing(w) ∧ r_agent(f,b) ∧ r_part(b,w))
FOLwDef. / **FE:** any Bird is *r_agent of* a Flight and has for *r_part* a Wing.
FOLwDef. / **FL:** Bird *r_agent* of : 1..* Flight, *r_part* : 1..* Wing;
FOLwDef. / **KIF:** (defrelation Bird (?b) := (exists ((?f Flight)(?w Wing)) (and (*r_agent* ?f ?b) (*r_part* ?b ?w))))
OWL2QL / **OWL_Functional-style:** SubClassOf (Bird ObjectIntersectionOf (ObjectSomeValuesFrom (*:r_agent* :Flight) ObjectSomeValuesFrom (*:r_part* :Wing))
RDF+OWL2QL / **Turtle:** Bird owl:subClassOf [*rdf:type* owl:Restriction; owl:onProperty *r_agent*; owl:someValuesFrom Flight]. Bird owl:subClassOf [*rdf:type* owl:Restriction; owl:onProperty *r_part*; owl:someValuesFrom Wing].
UML_model / **UML_concise_notation:**    **About this notation.** The arrow "−>" is for a super-type (subClassOf) link. For other links, the arrow "→" is used with an associated link type and a destination cardinality when this one is different from 0..*, i.e., 0–N. For *concision* purposes, boxes around classes (types) and associations (links) are not drawn.

Table 2 uses a more *precise* statement, still *derived* from the traditional "(All) birds fly" example. Indeed, not *all* birds *can* fly nor *constantly* fly. Some notations such as FE can directly represent complex statements. For other ones, when some constructs are missing – e.g., in Turtle, those for contextualization and generalized quantification – link types from IKLmE can be used as illustrated in Table 2: see the links of type r_ctxt, r_value or r_quantifier. Via RDF reification and named graphs (Carroll et al, 2005), some W3C models support certain kinds of meta-statements but none of them support "contexts" as above defined.

Table 2: Representation of a relatively *precise* statement, using contexts and a numerical quantifier, with a high-level notation and then a lower-level one.

English: On March 21st 2016, John Doe believed that in 2015 and in USA, at least 78% of adult healthy carinate birds were able to fly.
IKLmE / **FE:** ` ` ` ` ` `at least 78% of Adult_healthy_carinate_bird can be *r_agent* of a flight´ *r_place* USA´ *r_time* 2015´ *r_believer* JohnDoe´ *r_time* 2016-03-21´.
IKLmE / **Turtle:** [*r_value* [*r_value* [*r_value* [*r_value* [*r_value* [*r_quantifier* 78to100pc; *r_type* Adult_healthy_carinate_bird; *r_agent_of* [a flight]]; *r_ctxt* [*r_modality* Physical_possibility]]; *r_ctxt* [*r_place* USA]]; *r_ctxt* [*r_time* 2015]]; *r_ctxt* [*r_believer* JohnDoe]]; *r_ctxt* [*r_time* 2016-03-21]] .

3. Principles of our approach

3.1 Distinguishing and relating potentially useful top-level KRL element types

Existing KRLs were designed with many different viewpoints and assumptions. The top-level of KRLO organizes types useful for representing KRLs into a well-integrated whole. It already includes over 900 types, organized via alternative or complementary partitions and other semantic relationships.

Each KR being a description of something, and each description having a “semantic content” and an “instrument”, KRLO first distinguishes and relates types for these notions. For uniformization purposes, KRLO only offers types for description instruments plus one relation type to link them to their description content counterpart types. The distinction between description content and description instrument is not made in other ontologies of KRLs. However, SBVR – the ontology of the OMG (Object Management Group; omg.org) for “Semantics of Business Vocabulary and Business Rules” (omg.org/spec/SBVR/1.0) – has some types specific to each of these categories.

Then, KRLO represents types for languages, i.e., models and/or notations, and for language elements. It distinguishes abstract elements (AEs) from concrete elements (CEs). A CE is a notation element, e.g., an infix or prefix representation for an AE. An AE may be i) a formula, i.e., something denoting a fact, ii) an abstract term – i.e., something denoting a logic object – e.g., a value, a variable and a function call, or iii) a symbol, e.g., one for a quantifier, variable or constant. References – e.g., variables and function calls – are defined to be usable only where the AEs they refer to can be used. Each particular model is a selection and specialization of some of these AE types. E.g., models not allowing meta-statements do not include types for references to formulas. This specialization based approach enables people to organize types for models, notations and AEs, and to compare them. This also reduces the amount of specifications to write.

3.2 Representing abstract structures in a uniform function-like way

In KRLO, for uniformization purposes, the structure of an AE is represented like the structure of a function, i.e., as an operator with an optional set of arguments and a result. Hence, in KRLO, the *most important relation types usable as primitives for linking AEs* are named *r_operator, r_argument* and *r_result*. The first two are subtypes of *r_part*.

Thus, the structure of a relation is defined to include as *operator* a relation type, as *arguments* a list of AEs and as *result* a boolean. E.g., the structure for the relation “*r_part* (USA Iowa)” has as *operator r_part*, as *arguments* USA and Iowa, and as *result* True.

Similarly, the *structure* of a quantification statement is defined to include as *operator* a quantifier, some *arguments* and as *result* a boolean. A function is defined to have a type as *operator*, some *arguments* and a *result*. A variable or an identifier is (partially) defined to have as *operator* a name, as *arguments* an empty list of AEs and as *result* an AE of a certain type. Being an *operator* is only a structural role that different kinds of elements may have. Representing the structure of a formula is not asserting it. In RDF, this is the difference between asserting a statement and reifying it.

Euzenat and Stuckenschmidt (2003) view “translating between AEs” as applying one or several procedural or declarative “translation operators”, each satisfying a “translation property”. In KRLO, each AE type definition relating it to another AE type by a logic “equivalence” or “implication” link can be exploited as a translation operator: the translation properties are the logic “equivalence” or “implication”. E.g., KRLO currently proposes equivalences or implications i) between non-binary relations and binary ones, ii) between different structures for meta-statements, and iii) between some kinds of definitions and some uses of universal quantification with implication or equivalence relations.

3.3 Representing concrete specifications using functions, not predefined types

In KRLO the structure of each AE is known. Thence, the presentation of a given AE in a given notation can be specified by composing the presentation of each part of this AE. For textual notations, this composition is often a simple ordering – e.g., in a prefix, infix or postfix way – plus syntactic sugar to delimit some parts. E.g., assume that each AE “?ae” of type “aet” has an *operator* “o” of type “ot” and two *arguments* “x” and “y” respectively of types “xt” and “yt”, and that for such AEs their CEs in the notation “nt” are the form “ot [x ; y]”. Then, with KRLO and FL, the specification for such CEs is:

```
aet ?ae rc_spec: fc_spec_( List( fc_OP_from_(?ae,nt), "[", fc_ARGS_from_(?ae,nt,";"), "]" ),
                           nt, "");
```

The “c” in the above names means “concrete”. Links of type *rc_spec* relate AEs to specifications of their concrete presentation. Function names begin by “f”. Functions are used for concision. Functional relations could be used instead. fc_spec returns a specification for the presentation of an AE in the notation given in its second parameter, i.e., “nt” in the above example. fc_spec concatenates sub-specifications, e.g., those returned by fc_OP_from and fc_ARGS_from. fc_OP_from returns a presentation specification for the *operator* of its first parameter. fc_ARGS_from does the same but for the *arguments* of its first parameter. Since these arguments constitute a list, the first parameter of both fc_spec and fc_ARGS_from refers to a list. The third parameter of these two functions specifies the separators to use between the presentation of the elements in this list. Optional spaces are the default (hence the "") and “;” indicates that “;” must be used as separator in addition to optional spaces.

Thus, such presentation specifications may be as recursive and flexible as concrete grammars, in addition to being based on AEs and semantic links between them. As an example of flexibility, it is possible to specify that the presentation of some AE parts should be optional or omitted.

Inheritance is exploited: a presentation specification is associated to an AE type only if the specification provided by its supertypes is not sufficient. This implies that each specification is a default one: the destination of rc_spec has the numerical quantifier or cardinality “0..1”. Thanks to default presentation specifications in the top-level of OKRL, inheritance provides at least one presentation specification for each AE. For each notation, there should not be two direct *rc_spec* links from AEs of a same type: this relationship is functional. Instead, to specify two alternative kinds of presentations for the AEs of a type “t”, two subtypes of “t” must be made explicit and each one associated to a different kind of presentation. Finally, subtyping two types that would provide two *alternative* specifications must be prevented. Our tools check all this.

In the first version of KRLO, instead of using the above described presentation specification method, we began to create and exploit an ontology for the main prefixed, infixed or post-fixed forms that can be used for CEs. The resulting set of “predefined types” for CEs proved unmanageable: there were too many “type names to understand” and “parameters to use” for people to easily read or write such specifications.

3.4 Defining types of KR importing and exporting generic methods

The above described *uniform* representation of AEs *and* CEs enables the creation of *generic* methods for importing or exporting KRs. Both kinds of methods take at least a notation type as parameter. KRLO includes some specifications for such methods, along with types for specifying default options, e.g., regarding spaces and indentation. We implemented such methods in Javascript.

Here is the outline of the default generic export method in KRLO. Given an AE, a target model, a target notation and, optionally, a set of translation properties, the method

generates a CE for this AE by i) recursively navigating the parts of that AE, ii) selecting and applying the relevant AE translation operators on each of these parts *when* the target model requires it and if KRLO provides the links to do so, iii) accessing and applying *the* relevant presentation specification associated to each translated or original part, and iv) concatenating the resulting CEs. This export method satisfies the given translation properties: equivalence or implication. It is *complete with respect to what is expressed by KRLO* since there is one *rc_spec* relation for each AE in a given type of notation.
KRLO is only for purely declarative KRLs: it does not represent nor take into account any inference strategy. E.g., the default export method may generate rules in an order that leads to infinite loops if they are used with a Prolog inference engine.

4. Validation, use cases and comparisons

4.1 Validation by round-tripping

KRLO and our tools are validated together by ensuring that, for each KRL represented in KRLO, translating KRs to any other KRL of KRLO (complemented by IKLmE if necessary) and translating back the results lead to the original KRs, except for optional white space. For each KRL, the first test is actually simpler: importing and exporting a file of KRs must lead to the same file, except for optional white space. “Each KRL” means “each possible pair of model and notation”. The first input file is generated to include at least an instance of each AE type of the tested KRL. Then, the other tests use real-world files in this KRL, especially files from major ontology repositories, e.g., Ontohub (ontohub.org), DBpedia (dbpedia.org), Wikidata (wikidata.org) and YAGO3 (datahub.io/dataset/yago). Finally, KRLO itself is translated from FL to the target KRL – complemented by IKLmE if necessary – and back.
Currently, given the implementation stage of our tools, we have not tested all the features of all represented KRLs, only those related to the pairing of the FL, Turtle and JSON-LD notations with RDF+OWL2Full or IKLmE models. However, more features are actively implemented.

4.2. Use cases and first comparisons

The next three sub-sections illustrate the kinds of use cases that our approach for importing and exporting KRs can better support than other approaches since i) it relies on an ontology, ii) this one *homogeneously* represents and relates KRL models, and iii) this ontology includes specifications for notations.

4.2.1 Easing the implementation of tools exploiting KRs written in many KRLs

In WebKB-2, using C++ and a Lex&Yacc-like parser generator, we implemented KR import and export features for several KRLs. For each new KRL, this took several months owing to a lot of hard-to-factor new code and the combinations of an ever increasing number of options. Thus, this was not a scalable approach.
Such implementation tasks can be eased by a *programming environment* generator like Centaur (Borras et al, 1988) or, for KR translation purpose, the use of rule based languages such as those of ODE (Corcho, 2004) or QVT (Query/View/Transformation; omg.org/spec/QVT). E.g., Centaur proposed declarative languages for specifying concrete grammars, abstract grammars and rules mapping them. Based on them, Centaur could generate structured editors, parsers, type checkers, interpreters, compilers and translators. However, the above cited rule based languages and those of Centaur are low-level and execution oriented, not modeling oriented: they do not ease the creation and exploitation of ontologies (Guizzardi, 2010). Thus, with them, as with procedural code, i) small changes in KRL features often lead to important changes in the code, and ii) translations use the *direct mapping* approach, i.e., from one language to another, between AEs or CEs. This is still not a scalable approach for handling many KRLs.

4.2.2 Enabling end-users of tools to adapt KRLs or the presentation of KRs
Even though we designed FL to be high-level, when modeling knowledge, we often had to extend it for creating or presenting high-level KRs and thus also ease our task. End-users of tools other than our own can do so only in limited ways. E.g., even though the Model Driven Engineering tool BAM (Feja et al, 2011) was designed to handle several KRL models and notations, its meta-model is predefined, not ontology-based; hence, for model and notation extension purposes, BAM only proposes macros and informal annotations.
End-users of tools other than our own also cannot *adapt* a KRL to ad-hoc ways it is used. Yet, if KRs created by a person or generated by a tool systematically include a same incorrect or ad hoc usage pattern with a certain KRL, the specification of this KRL can often be extended for the pattern to be correctly interpreted. Such extensions would enable people to exploit or better exploit existing resources. E.g., Beek et al (2014) reported that only 37.6% of Datahub resources for Linked Data are fully machine-processable.
Several tools propose *rule based and/or style-sheet based* transformation languages *for RDF*, e.g., SPARQL Template (Corby and Faron-Zucker, 2014). These tools enable their users to specify how RDF AEs may be presented, e.g., in a certain notation, in a certain order, in bold, in a pop-up window, etc. However, these tools and languages were not meant to use ontologies for models or notations: they are RDF-based and require the use of a new style-sheet for each target notation.
It seems that no ontology other than KRLO includes specifications for notations, hence that no tool other than our own exploit such specifications. Most current KRL translators are for notations usually associated to RDF or OWL, e.g., EasyRDF (easyrdf.org/converter) and those based on the OWL API. Unsurprisingly, their Web interfaces or APIs propose very few options to parameter their import and export of KRs.

4.2.3 Comparing and evaluating KRLs or KRs according to criteria or best practices
This requires a homogeneous ontology of KRLs. Evaluating the structure of KRs also requires a tool that can parse them and i) perform commands on AEs for these KRs, or ii) export these AEs, i.e., their structures, in a given target KRL. This last case, which is possible too with our tools, enables the use of another inference engine to check KR structure related criteria or best practices. This may for example be useful for selecting KR resources or checking the skills of students in KM.

4.4 Other works based on an ontology of KRLs
The main ones are described below. Unlike KRLO, the ontologies of such works are only about AEs of one or few particular models, and do not represent AE structures in a uniform way. They were only meant for translating models, not notations. Ontolingua (Gruber, 1993) also was for translation purpose but did not represent models, even though it included definitions for types similar to those of OWL2.
Euzenat and Stuckenschmidt (2003) used XSLT (the Extensible Stylesheet Language for Transforming XML documents; w3.org/TR/xslt) to implement about 40 translation operators or relations between AEs from 25 description logics.
The LATIN (Logic Atlas and Integrator) Project (2009-2012) (Codescu et al, 2001) represented translation relations between different logics. LATIN has been exploited via HETS (Heterogeneous Tool Set; hets.eu) or MMT (Meta-Meta-Theory; uniformal.github.io). Both HETS and MMT handle various logics and inference engines for reasoning purposes, e.g., OWL and HOL logics. Neither export knowledge in KRLs different from the ones they were imported with.
Via DOL (Distributed Ontology modeling Language; omg.org/spec/DOL), the OMG proposes a meta-KRL for i) specifying particular kinds of translation relations between KRL models, and ii) using several KRLs in a same DOL document. HETS can handle DOL.
ODM 1.1 (Ontology Definition Metamodel; omg.org/spec/ODM), an OMG specification of 2014, uses UML for representing four KRL models: RDF, OWL, CL and Topic Maps. Between AEs of these models, it sets some semantic relations such as generalization or equivalence relations. It gives some *direct mappings* between AEs of different models but via QVT rules, hence not in ways that can be directly used by inference engines. Since direct mappings are used instead of *few types as primitives for defining and relating the various AEs*, the heterogeneity of the various models is not eliminated.

5. Conclusion

This article has shown that it is possible to specify *generic* “KR import or export methods” by specifying and exploiting a *homogeneous* ontology of KRL abstract models *and* notations such as KRLO. At a high-level, this article has also shown how this is possible and why this is interesting. Only declarative KRL translation was addressed: neither the content of KRs – i.e., their information not related to the used KRLs – nor particular inference engine methods were taken into account. No “minimal translation” between logics has been specified: IKLmE is used as a complement whenever the target KRL lacks some needed expressiveness. No method exploiting the “KR import or export methods” has yet been specified or implemented, e.g., no method for “structured editing of KRs”, “ontology design pattern checking” or “KRL expressiveness detection”. Finally, KRLO is currently focused on few KRLs. However, KRLO was designed for scalability and specifying or extending a model or a notation in KRLO does not require much more time than specifying or extending the notation of a KRL with a classic concrete grammar.
The company employing the second author of this article has begun incorporating the results of our work in some of its software products for them to i) collect and aggregate knowledge from knowledge bases, and ii) enable end-users to adapt input and output formats to their needs.
In the immediate future, we shall continue extending and validating KRLO and our tools. to handle more KRLs. The next step is to handle some KR query languages, starting with SPARQL, and to complement our notation ontology by a *presentation ontology* with concepts from style-sheets and user interfaces. Then, still for KR importing and exporting, we shall exploit KRLO with SPARQL and MMT.

References

Beek, W., Groth, P., Schlobach, S. and Hoekstra, R. (2014) “A web observatory for the machine processability of structured data on the web”, 2014 ACM conference on Web science, pp 249-250.

Borras, P., Clément, D., Despeyrouz, Th., Incerpi, J., Kahn, G., Lang, B. and Pascual, V. (1988) “CENTAUR: the system”, SIGSOFT'88 (Boston, USA), pp 14-24.

Carroll, J., Bizer, C., Hayes, P. and Stickler, P. (2005) “Named graphs, provenance and trust”, WWW 2005 (Chiba, Japan, 10-14 May 2005), pp 613-622.

Codescu, M., Horozal, F., Kohlhase, M. , Mossakowski, T. and Rabe, F. (2011) “Project Abstract: Logic Atlas and Integrator (LATIN)”, *Intelligent Computer Mathematics 2011*, LNCS 6824, pp 287-289. See also http://trac.omdoc.org/LATIN/

Corby, O. and Faron-Zucker, C. (2015) “STTL: A SPARQL-based Transformation Language for RDF”, WEBIST 2015 (Lisbon, Portugal).

Corcho, Ó. (2004). *A Layered Declarative Approach To Ontology Translation With Knowledge Preservation*, PhD Thesis (311 pages), Universidad Politécnica de Madrid.

Euzenat, J. and Stuckenschmidt, H. (2003) “The 'family of languages' approach to semantic interoperability”, *Knowledge transformation for the semantic web* (eds: Borys Omelayenko, Michel Klein), IOS press, pp 49-63.

Guizzardi, G., Lopes, M., Baião, F. and Falbo, R. (2010) “On the importance of truly ontological representation languages”, *IJISMD*, Vol 1, No. 2, pp 1-22.

Hayes, P.J. (2006) *IKL guide,* [online], www.ihmc.us/users/phayes/IKL/GUIDE/GUIDE.html

Horridge, M. and Bechhofer, S. (2011) “The OWL API: A Java API for OWL ontologies”, *Semantic Web Journal*, Vol 2, No. 1, January 2011, pp 11-21.

Martin, Ph. (2002) “Knowledge representation in CGLF, CGIF, KIF, Frame-CG and Formalized-English”, ICCS 2002, LNAI 2393, pp 77-91.

Martin, Ph. (2009) *Towards a collaboratively-built knowledge base of&for scalable knowledge sharing and retrieval,* HDR thesis (240p; “Habilitation to Direct Research”), University of La Réunion, France.

Sowa, J.F. (1984) *Conceptual Structures: Information Processing in Mind and Machine*, Addison-Wesley, Reading, MA, 1984.