Top-level Ideas about Importing, Translating and Exporting Knowledge via an Ontology of Representation Languages

Philippe MARTIN
EA2525 LIM, ESIROI, Uni. of La Réunion,
F-97490 Sainte Clotilde, France, +262 262 48 33 30
+ adjunct researcher of the School of I.C.T.
at Griffith University, Australia
Philippe.Martin@univ-reunion.fr

Jérémy BÉNARD
GTH, Logicells, 55 rue Labourdonnais,
97400 Saint Denis, France
+262 262 20 93 85
Jeremy.Benard@logicells.com

ABSTRACT
This article introduces KRLO, an ontology of knowledge representation languages (KRLs), the first to represent KRL abstract models in a uniform way and the first to represent KRL notations, i.e., concrete models. Thus, KRLO can help design tools handling many KRLs and letting their end-users design or adapt KRLs. KRLO also represent KRL import, translation and export methods in a declarative way, both via Datalog like rules and pure functions.

CCS Concepts
• Artificial Intelligence→Knowledge Representation Formalisms and Methods • Representation languages

Keywords
Knowledge Representation Languages (KRLs); Ontology of KRLs; Knowledge Integration; Language Technologies.

1. INTRODUCTION

KRLs are *languages* for representing information into logic-based forms – *knowledge representations (KRs*) – within knowledge bases (KBs). They permit logical inference engines and precision-oriented information sharing (Linked Data,...). The "Semantic Web" is the set of representations that use KRLs from the W3C.

Many KRLs exist. A unique one would not be adequate for every kind of knowledge modelling or exploitation, nor for every person or tool. E.g., poorly expressive KRLs may have good computational properties but may lead people not to represent some knowledge or represent it in biased ways. Many KBs use more expressive KRLs. Many applications require handling many KRLs. For knowledge entering, reuse and interoperability purposes, *importing, exporting or translating KRs expressed in different KRLs* is needed.

KRLs may have *abstract models*, e.g., the W3C Resource Description Framework (RDF), the W3C OWL2 Web Ontology Language model, and Common Logics (CL), the ANSI standard for KRLs based on First-Order Logics (FOL). These are abstract *data structure* models, such as the models or meta-models of Model Driven Engineering (MDE). These are *not* theory models of model-theoretic semantics. Different abstract models may follow different logics, e.g., FOL or a description logic. An abstract model may be *formally presented* via different *notations*, i.e., *concrete models or* concrete syntaxes, e.g., the CL Interchange Format (CLIF), Turtle or XML-based notations. From now on, unless preceded by "concrete", "model" refers to an abstract model. Models and notations *are* themselves KRLs: a KRL *is* a model and/or a notation. In this article, an *element* is a KRL element. A *concrete element* (CE) is a notation element, e.g., an infix or prefix representation, as in "3 = 2 + 1" and "= (3 +(2 1))" or "(= 3 (+ 2 1))". An *abstract element* (AE) is an element of a model. A model is a set of AEs. 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 variable or function call, or iii) a *symbol*, e.g., one for a quantifier, variable or constant.

Importing KRs is done by a syntactic parser which generates CEs (e.g., organized into a Concrete Syntax Tree) and/or a semantic analyser which generates AEs, e.g., organized into an Abstract Syntax Tree (AST) or an Abstract Semantic Graph (ASG). If these AEs are not the ones required by the importing tool, a translation to other AEs occurs. When, as with XML based notations, the input notations are homo-iconic, i.e., when the structure of the CEs mirrors the structure of the AEs, the parser may also directly be a semantic analyser.

Exporting KRs expressed in a KRL goes in the reverse direction.

Translating KRs is translating their *logic or non-logic* objects, i.e, the KRLs and the KRs content. It may be directly between CEs. More flexibility and genericity is achieved when AE translation is involved, i.e., when the import, translation and export processes are separated. Thus, current research works on translation focus on translation between AEs. However, they assume that the import and export processes are done separately, via other techniques.

This article is not about *KR content* translation. It is about the *KRL related part* of importing, translating and exporting KRs. The advantages of our approach first come from its exploitation of *an ontology of KRLs* in each of the last three processes. These advantages also come from the *three originalities* of the particular exploited ontology of KRLs. We named it KRLO. It is the first to represent KRL abstract models of different families in a *uniform way*, e.g., the RDF+OWL models and the CL model. KRLO is also the first to include an ontology of KRL notations. Finally, KRLO is the first to include rules and functions specifying default methods for input, translation and export purposes. Thus, this article is also about this ontology. We have designed some tools to help exploit it.

SEMANTiCS '16, September 12-15, 2016, Leipzig, Germany.

DOI: http://dx.doi.org/10.1145/2993318.2993344

Our approach, KRLO and these tools are meant to ease the implementation of applications handling many KRLs, including KRLs specified by end-users. These tools are: a generic *KRL parser and semantic analyser*, a *KRL translation Web server* based on it, and a Web server allowing its users to complete KRLO, for example with new models and notations. This last server directly reuses our shared KB server WebKB. Its KB sharing protocols have already been published [9] and hence will not be presented in this article. KRLO, these servers, and a very extended version of this article, are accessible from http://www.webkb.org/KRLs/.

Section 2 gives uses cases for our approach and situates it with respect to other ones. Section 3 gives some principles for the specification of KRLs in or via KRLO, and for its default inport, translation and export rules or functions.

2. COMPARISONS AND USE CASES

2.1 Approaches not based on ontologies

Importing and exporting KRs are often implemented procedurally, hence in long and error prone manual ways, using *parser generators*, such as Lex&Yacc, or via application programming interfaces (API) such as the Open Knowledge Base Connectivity (OKBC).

Programming environment generators are designed to ease such tasks, including in knowledge engineering [4]. E.g., Centaur [2] proposed declarative languages for specifying concrete grammars, abstract grammars and rules bridging them. Based on them, it could generate structured editors, parsers, type checkers, interpreters, compilers and translators. However, being Prolog-like, *these declarative languages are execution oriented*, not modelling oriented: they do not ease the creation and reuse of ontologies with given or calculated subsumption and exclusion relations [6] [8]. Thus, with them, as with procedural code, i) small changes in the KRLs to specify often lead to important changes in the specifications, and ii) the specifications are difficult to organize into an ontology and hence are not as easy to compare, automatically analyze and reuse as in an ontology based approach. Regarding translations, Centaur is also limited since it uses the *direct mapping* approach, from one language to another. With Centaur, the KRL designers provides translation rules between AEs. To translate between its KRLs, the W3C provides direct mapping rules between CEs, e.g., i) from the OWL2 Functional-Style Syntax (FSS) into CEs of the RIF-PS notation for the RIF-BLD model, and ii) from CEs of FSS into CEs of Triples notation for the RDF model. RIF is the W3C Rule Interchange Format. RIF-BLD is its Basic Logic Dialect, the one for definite Horn rules with equality, i.e., for Datalog like KRLs. RIF-PS is the Presentation Syntax of RIF. The W3C does not propose translations between AEs. Indeed, i) RIF-BLD is not represented into an ontology, and ii) CEs of FSS directly represent AEs of the OWL2 Structural Specification. CL and RIF permit a *pivot* approach for KRL translations: they are intermediary expressive KRLs from and to which other KRLs can be translated. They are KRL pivots. KRLO can be seen as an “ontology of KRLs” pivot.

Languages created via *structure description* languages such as XML, MOF – the Meta-Object Facility of the *OMG* (Object Management Group) – or those used in Model Driven Engineering (MDE) are easy to parse and check via rather generic tools. E.g., XML tools also work on RDF/XML. However, i) these “rather generic tools” do not perform logical inferences, and ii) the concrete descriptions they exploit are often not concise or high-level enough to be used directly for knowledge entering/display or by tools for knowledge handling. E.g., which inference engine uses XML objects internally? Translations from/to other models or notations are still necessary. Our approach provides an ontology-based concise alternative to the use of XML as a meta-language for creating KRLs that follow given KRL ontologies. Thus, any notation can be used for the specifications. Languages such as Mof2Text, XSLT and CSS are difficult to use for KR presentation specifications since they are not KRLs or KR query languages. This is why there have been several works on *rule based and/or style-sheet based* transformation languages *for RDF*. They specify how RDF AEs may be presented, e.g., in a certain notation, in bold, in a pop-up window, etc. Examples of such tools are Xenon, Fresnel, OWL-PL and SPARQL Template. These tools were not initially meant to use a notation ontology: they initially required the use of a new style-sheet for each target notation. However, some of these tools – e.g., SPARQL Template – could exploit KRLO.

Thus, a first kind of use case for an approach based on an ontology like KRLO is to *ease the implementation of tools that parse, check translate or generate KRs in many KRLs*. This can be done by adding KRL specifications and using or adapting the default functions or rules of KRLO for KR import, translation and export.

A related second kind of use case is to *permit such tools to enable their users – i.e., each application developer or each end-user – to extend or adapt KRLs and the default import, translation or export methods* in advanced ways. With a procedural implementation or the use of a *fixed* (meta-)model for KRLs, this is hardly possible: only macros or informal annotations can be proposed. Without an ontology of KRL model and notation, this is difficult. In many cases, implementing or extending a KRL parser, translator, displayer or navigator is not an option or a cumbersome one. Yet, it is also often interesting to add syntactic sugar or new structures to a particular notation, to gain conciseness, ease readability or lift some expressiveness restrictions. Nowadays, when people want to represent knowledge that cannot be fully expressed with the KRLs they need or wish to use, they represent the knowledge in incomplete or ad hoc and biased ways. Instead, with our approach they can extend their KRLs. Conversely, our approach also provides a way to exploit KBs even if they include syntactically or semantically incorrect KRs, as long as the kinds of errors are systematic. Indeed, with our approach, the used KRL model, notation or parser can be adapted to interpret some *systematic* incorrect usages in special ways. This is useful: in the study of [1], only 37.6% of *Datahub resources for Linked Data* proved fully machine-processable. Similarly, KRLO could be extended for generating *Semantic Web wrappers*, i.e., tools parsing structures in certain resources to extract KRs.

2.2 Approaches based on ontologies

Except for KRLO, ontologies about KRLs are only about the AEs of some particular models. They were only meant for translation purposes. [7] showed that translation approaches (mapping, pivot, ...) are generalized when an ontology of KRL models is used, one where AEs – and then models – are related by certain *translation relations*. Each of these relations has an associated definition for performing the translation. Each relation also represents *translation properties*, i.e., whether or not the translation preserves the model-theoretic semantics, interpretations, and logic consequences of the translated AEs. Thus, given AEs to translate, a tool exploiting this approach may allow its users to choose different target models according to what they want the translations to preserve. As a proof-of-concept, [7] used XSLT to implement about 40 translation relations between AEs from 25 description logics. The LATIN (Logic Atlas and Integrator) Project (2009-2012) [3] went further by representing such translation relations between many different logics. Via HETS (Heterogeneous Tool Set), LATIN exploits various KRLs, including KRLs of HOL (Higher Order Logic) expressiveness, e.g., Isabelle and HasCASL. Via DOL (Distributed

Ontology modeling Language) [5], the OMG proposes a standard KRL for i) specifying particular kinds of translation relations between KRL models and ii) using several KRLs in a same DOL document. DOL is also implemented via HETS and the authors of DOL see some results of LATIN as avenues for future extensions [5]. Ontohub is a DOL based repository which includes KRL models and translation relations between them. DOL and HETS do not specify notations. They rely on external parsers and exporters.

None of the above cited works specifies or exploits an ontology representing AEs of different KRL models in a uniform and organized way. KRLO does so by linking the AEs with as many generalization relations and partOf relations as possible and, for these last ones, using the *operator-arguments schema* detailed in Section 3. Thus, KRLO can be seen or used as a way to align and integrate other KRL translation related ontologies. In KRLO, when direct relations cannot be used for defining AEs, CEs or inport/translation/export methods, rules *and* pure functions are used. Both are used in order to increase reusability. The functions use the representations as data structures. So far, in KRLO, the maximum required expressiveness is OWL2-DL plus rules allowing existential quantifiers in their heads. Since the structures in KRLO are akin to reified statements, the expressiveness required for their translation is unrelated to the one needed to express the content of the KRs. Translations represented *via rules* in KRLO are semantic preserving *structure translation rules*. For KR content translations, KRLO users have to exploit complementary ontologies and, possibly, more powerful inference engines. *In the future*, KRLO will be added to Ontohub and made exploitable via HETS.

None of the above cited works specifies or exploits an ontology of notations. KRLO does. This permits the import, translation and export tasks to exploit an ontology of KRLs and the same one. Thus, our approach extends the one presented in [7] and may be seen a *pivot* approach based on such an ontology of KRLs instead of a KRL.

Besides KRLO and Ontohub, there is another *ontology relating KRL models not belonging to a same family*: ODM 1.1, an OMG specification of 2014. It uses UML for representing four KRL models: RDF, OWL, CL and Topic Maps. Between AEs of these models, it also sets some semantic relations such as generalization or equivalence relations. Finally, it gives QVT rules for direct mappings between AEs of different models. Since direct mappings are used instead of *few primitives for defining and relating the various AEs*, the heterogeneity of the various KRL models is not eliminated. This heterogeneity makes the AEs difficult to *compare* or e*xploit in a uniform way*. Finally, QVT rules are not directly usable by inference engines and translating them into KRLs may not be easy. We are not aware of works using ODM for KR import, translation or export purposes. Similarly, we have not found *ontologies for notations*, even for RDF notations. Hence, apart from our KRLO based translator, it seems there is no other translator based on an ontology for a KRL model and a notation. There are translators between notations for the RDF model, e.g., RDF-translator, EasyRDF and RDF2RDF. Their Web interfaces or APIs propose no way to parameter their knowledge import, translation and export processes.

To sum up, regarding KRL translation, a kind of use case for our approach is an increased simplification of the *implementation of this process*. A related kind of use case is to enable end users to *parameter*, *extend or adapt* KRL translations. Adding a new KRL specification to KRLO – e.g., via its Web servers – may be sufficient to specify the structural translation of this KRL to other represented KRLs: no transformation rule or function may be required. Finally, as detailed by the authors of DOL, LATIN and [7], it is easier to implement a tool generating proofs for the properties of its translations if the tool exploits rules or relations in an ontology.

3. DESIGN PRINCIPLES OF KRLO

The top-level of KRLO, i.e., the part reused and specialized by specifications for particular KRLs, currently includes over 900 AE types. The structure of *each* KRL element is represented in a uniform way, like the structure of a function, i.e, as an operator with an optional set of arguments and a result. Thus, in KRLO, the six *most important primitive relation types for relating AEs* are named has_operator, has_argument, has_arguments, has_result, has_parts and has_part. The first two are subtypes of the last since the operator and arguments of an AE are also its parts. 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 "has_part (USA Iowa)" has as operator has_part, as arguments USA and Iowa, and as result True. The structure of a quantification 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. Thus, in KRLO, an AE operator may be a function/relation/collection type, a quantifier or a value. Being an operator is only a structural role that different kinds of elements may have. When the AE is a formula, representing its structure is not representing its meaning directly. In RDF, this is illustrated by the difference between a statement and its reification.

A CE is the presentation of an AE in a given notation. Since the structure of each AE is known, the presentation of CEs for a given AE can be specified by composition of the presentation of CEs for parts of this given AE. For textual notations, this composition is often a simple ordering – e.g., in a prefix, infix or postfix way – plus some syntactic sugar to delimit some parts. Thus, like the structure of an AE, the structure of a CE can be represented like the structure of a function. This led us to note that, like models, notations can be represented in a uniform way using few primitives. This also led us to note that a *generic* analyser could be built for all KRLs that can have a deterministic context-free grammar, e.g., an LL(1) or LALR(1) grammar. Furthermore, this analyser would be efficient since these grammars that can be efficiently parsed. As an example for the underlying idea, consider an AE composed of an operator "o" with two arguments "x" and "y". If single parenthesis are mandatory delimiters and if single space characters are the only usable separators, this AE has only the next five possible CEs *in all the notations we know*: "o (x y)", the prefix functional form, as in RIF-PS; "(o x y)", the prefix list-like form, as in KIF; "(x o y)", the infix form, as in Turtle and some RIF-PS formulas; "(x y o)", the postfix list-like form; "(x y) o", the postfix functional form. Five rules of an LL(1) or LALR(1) grammar can be used for specifying these five forms and they can be generalized for any number of arguments, not just two. Furthermore, if – as with the Lex&Yacc parser generators – the grammar can be divided into a lexical grammar and a non-lexical grammar, the separators can be made generic via terminal symbols with names such as Placeholder_for_begin-mark_of_arguments_of_a_prefix-function-like_element. Based on these ideas and using Flex&Bison, variants of Lex&Yacc, we created a *generic analyser for any KRL that can have an LALR(1) grammar*. This grammar for this KRL does not have to be found since it is generalized by the generic grammar that our analyser uses. Given a CE and its KRL, based on the specifications for the notation and model of this KRL in KRLO, this analyser directly generates an *abstract semantic graph*. Its methods are the default input methods of KRLO. They are represented via functions and, soon, rules too. From these functions, Javascript functions are generated to implement the analyser in Javascript. This analyser is,

as most generated parsers, based on a finite state machine, not on top-down direct interpretation rules, as often done in Prolog [10].

The top-level of the *ontology of notations of KRLO* does not categorize all possible prefix/infix/postfix notation forms for an AE. Instead, it contains primitive relations permitting to describe them and, for usability purposes, functions to combine them. They permit to specify i) the structure of CEs for a given type of AE in a given type of notation, and ii) the order and roles of these CEs: operator, argument or separator. Thanks to these roles, relevant grammar rules can be selected to be executed, e.g., rules belonging to the grammar of our generic analyser. For exporting into textual CEs, only the order of the terms in this list is important. The spacing between CEs, e.g., for indentation purposes, is similarly defined.

In KRLO, each AE has *one* and *only one* rc_spec relation for a given type of notation, i.e., one CE specification. This relation may be inherited or overriding an inherited one. The *"only one"* part is ensured by our knowledge server which prevents the entering of ambiguities when it detects them. The other *"one"* part comes from the default presentation specifications of KRLO, e.g., for the spacing between CEs. Thus, for a given AE and notation, all possible CEs are unambiguously described. This is why the import and export methods of KRLO can be represented not only as rules but also as functions or as functional relations for the KRLs that do not handle functions.

Given an AE and specifications related to a target KRL, the default *export method* specified in KRLO generates a CE for this AE by i) recursively navigating the parts of that AE, ii) translating each of these parts *when* the target model requires it and *when*, to do so, the method can find relations, rules or functions that satisfy the *translation properties* selected by the user (as noted in Section 2.2), iii) for each translated or original part, applying *the* CE specification associated to this part in the target KRL specification, and iv) concatenating the resulting CEs. The translation and export processes are *complete with respect to what is expressed by these relations, rules or functions* since there is one rc_spec relation for each AE in a given type of notation. Each user can *control* the translation and export processes. Indeed, she can *select* not only the translation properties but also the target notation and the target model or the target AEs. She can also *extend* KRLO or *adapt* her copy of KRLO. In the general case, knowledge export and translation are arbitrary in the sense that knowledge can be translated and exported in various ways. However, this is not the case with KRLO in the sense that *particular ways* can be represented and then selected by the user. The default presentation choices represented in KRLO permit the user not to do this work if she does not want to. However, KRLO does not represent *all* information necessary for *any* export or translation to be semantically complete with respect to *any* application. E.g., KRLO does not yet represent any inference strategy and hence the order of statements generated by translation and export processes may not be adequate for a particular inference strategy. As an example, rules may be generated in an order that lead to infinite loops if they are used with a Prolog inference engine.

For KRL translation, in addition to equivalence or implication relations between types of AE, KRLO currently proposes some equivalences or implications via functions and rules. These rules and functions are for *translations between structures*, e.g., between i) non-binary relations and binary ones, ii) different structures for meta-statements (formulas about formulas), and iii) some kinds of definitions and some uses of universal quantification with implication or equivalence relations. These *structural translations* are simple: they can generally be expressed via RIF-BLD rules and do not require the complex strategies of general *term/graph rewriting techniques*. Backward chaining is sufficient to exploit them. Thus, KRLO does not specify a default *translation method* for combining these translation rules or functions.

For translations not yet supported by KRLO, the user has to import complementary ontologies. KRLO does not define types which are not logic-related, e.g., types for physical quantities or dimensions. Thus, *if* a KRL notation has some syntactic sugar for such types, the notation specification has to reuse types that are not defined in KRLO but in other ontologies. Using them for translation may require special translation methods.

4. CONCLUSION

This article showed the *interest of exploiting an homogeneous ontology of KRL models and notations* when implementing tools importing, translating and exporting knowledge in many KRLs. It also showed *how we created such an ontology*, which main ideas we used, and which first tools we designed to help its exploitation. A related underlying contribution is the resulting ontology, KRLO, which i) represents and relates several models and notations into a unified framework, ii) declaratively specifies some import and export methods, and iii) can be extended with additional specifications by Web users. The approach also provides an ontology-based concise alternative to the use of XML as a meta-language for creating KRLs.

5. REFERENCES

[1] Beek, W., Groth, P., Schlobach, S. and Hoekstra, R. 2014. A web observatory for the machine processability of structured data on the web. In *Proceedings of the 2014 ACM conference on Web science*, 249-250.

[2] Borras, P., Clément, D., Despeyrouz, Th., Incerpi, J., Kahn, G., Lang, B. and Pascual, V. 1988. CENTAUR: the system. In *Proceedings of SIGSOFT'88, 3rd Annual Symposium on Software Development Environments* (Boston, USA), 14-24.

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

[4] Corby, O. and Dieng, R. 1996. Cokace: a Centaur-based environment for CommonKADS Conceptual Modeling Language. In *Proceedings of ECAI 1996* (Hungary), 418-422.

[5] DOL 2016. *The Distributed Ontology, Modeling, and Specification Language (DOL).* OMG document. http://www.omg.org/spec/DOL/

[6] Dromey G.R. (2006). Scaleable Formalization of Imperfect Knowledge. Proceedings of AWCVS-2006, 1st Asian Working Conference on Verified Software, Macao SAR, China.

[7] 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, 49-63.

[8] Guizzardi G., Lopes M., Baião F., Falbo R. On the importance of truly ontological representation languages. IJISMD 2010.

[9] Martin, Ph. 2011. Collaborative knowledge sharing and editing. *International Journal on Computer Science and Information Systems*, Vol. 6, Issue 1, 14-29.

[10] Wielemaker, J., Schreiber, G. and Wielinga, B., 2003. Prolog-Based Infrastructure for RDF: Scalability and Performance. In *Proceedings of ISWC 2003*, LNCS 2870, 644-658.