# Assembling Rule Mashups in the Semantic Web

Guillermo González-Moriyón[1], Luis Polo[1], Diego Berrueta[1], Carlos Tejo-Alonso[1], and Miguel Iglesias[2]

[1] Fundación CTIC
Gijón, Asturias, Spain
name.surname@fundacionctic.org
http://www.fundacionctic.org/
[2] ArcelorMittal
Avilés, Asturias, Spain
miguel.iglesias@arcelormittal.com
http://www.arcelormittal.com/

**Abstract.** This paper introduces RIF Assembler, a tool that reuses knowledge to automatically construct rule-based systems. Our novel approach is based on (a) annotating domain rules with metadata and (b) expressing assembly instructions as metarules that manipulate the annotations. We leverage the power of RIF as a rule interchange format, and of RDF and OWL as languages for rule annotations. RIF Assembler has applications in many scenarios. This paper presents two of them in increasing order of sophistication: a simplistic example related to the health care industry, and an actual usage in the steel industry that involves the construction of a decision-support system driven by business process descriptions.

**Keywords:** rules, RIF, metarules, rule mashups.

## 1 Introduction and Motivation

Rule-based systems are widely used to implement business applications within organizations. Rules capture knowledge in a declarative form. Complex logic can be assembled by combining small and understandable rules.

A category of software products known as BRMS (Business Rule Management Systems) aims to facilitate the development and maintenance of rule-based systems. The market for these tools is dominated by IBM, Oracle and Red Hat. Although they are feature-full, BRMSs are not silver bullets. The management of large collections of rules within an organization is still challenging (rule-based applications may contain thousands of rules). Furthermore, although rules are small, understandable and declarative, their reuse between systems is not always straightforward. Rules are application-specific and bound to a particular technology.

RIF aims to solve the latter issue by providing a common interchange format for rules [2]. RIF is a W3C standard that opens the door to a new scenario

E. Simperl et al. (Eds.): ESWC 2012, LNCS 7295, pp. 590–602, 2012.

in which behavioral knowledge can be exchanged using web standards, as a complement to the exchange of static models (OWL ontologies) and factual data (RDF descriptions). Nevertheless, to deal with the issue of the rules being application-specific, and to create rule mashups, it becomes necessary to select, adapt and modify them so they can fit into new roles. These transformations permit reuse of the rules across many applications.

RIF Assembler is an application conceived to automate and simplify the construction of rule-based systems that gather already-available knowledge pieces. It permits assembly of new rulesets by picking and selecting from the catalogue of existing rules, as well as adapting them to serve their new purpose if necessary. Due to its verbosity and complex grammar, the normative XML syntax of RIF (RIF/XML) does not offer a convenient platform to carry out these operations. Instead, our approach relies on manipulating rules as RDF resources for easy querying and transformation. Therefore, a bidirectional mapping between RIF/XML trees and RDF graphs has been defined.

The paper is organized as follows. The next Section introduces a (partial) bidirectional mapping between RIF documents and RDF graphs. Section 3 describes how RIF Assembler works. Two usage scenarios are discussed in Section 4. Related work is examined in Section 5 and, finally, concluding remarks are made in Section 6.

## 2 Moving between RIF and RDF

RIF is a family of languages, called *dialects*, covering different kinds of rules: from logic-programming [10] to production rules [5]. The syntax and semantics of each dialect is rigorously and formally specified, sharing a common core of machinery. Among their shared features, RIF dialects include support for annotations.

In RIF, annotations allow metadata to be attached to almost every syntactic element of the language, from the RIF document itself (top element in the hierarchy) to the terminal symbols of the grammar[1]. No more than one annotation is allowed per element. An annotation has the form `(* id` $\varphi$ `*)`, where `id` represents the identifier of the annotated syntactic element (a URI), and $\varphi$ is a RIF formula that captures the metadata. In particular, $\varphi$ is a frame (an expression of the form `s[p ->o]`) or a conjunction of frames (i.e., `And(`$s_1$`[`$p_1$` ->`$o_1$`], ... ,` $s_n$`[`$p_n$` ->`$o_n$`])`).

The RIF machinery for annotations is very flexible and permits great syntactic freedom. For instance, the identifier (`id`) is not required in general. However, when absent, it is not possible to attach metadata to the element nor can cross-references be made between elements of a RIF document. Thus, our advice to authors of RIF documents is to assign identifiers to at least groups and rules in order to facilitate reusage.

[1] RIF normative interchange syntax is an XML grammar, although for the sake of readability, in this paper we will use the more human-readable RIF Presentation Syntax (PS), defined in the informative part of the specification.

```
Group (
 (* ex:bf-rule  ex:bf-rule [
       dc:title    -> "Breast feeding contraindication alert for Nafarelin"
       dc:relation -> <http://www.drugbank.ca/drugs/DB00666>
       dc:source   -> <http://www.sometrustworthysite.com/>
    ]
 *)
 ?Person[ex:avoid-drug -> <http://www.drugbank.ca/drugs/DB00666>]
 :-
 ?Person # foaf:Person
 ?Person[foaf:gender -> "female"
         ex:status   -> ex:breastFeeding]
)
```

**Fig. 1.** A RIF rule with annotations

Figure 1 contains an example of a RIF rule in the domain of health care. Note that the metadata make use of existing ontologies and vocabularies, such as Dublin Core, FOAF and domain ontologies.

### 2.1 From RIF Documents to RDF Graphs

The following paragraphs define a translation from RIF documents to RDF graphs. The translations of the annotations and the representation of the hierarchical structure of rules and groups (groups are also known as rulesets) are treated separately.

Table 1 describes a mapping ($\pi$) between RIF metadata expressions ($\varphi$) and RDF triples. For the sake of simplicity, base terms in $\varphi$ are limited to constants (i.e., variables and other terms, such as positional terms, are not permitted). This constraint makes the translation straightforward because both ends use URIs and literals for constants.

It is worth noting the divergence between annotations translated by $\pi$ and the RDF syntax for RIF suggested by W3C [8]. In our case, semantically-equivalent triples for $\varphi$ annotations are provided according to [4] (i.e., there is a direct correspondence between a frame and a triple), while [8] describes an RDF-serialization for its frame-based syntax. Although the latter is more expressive and permits representation of complete RIF documents, it is more difficult to query using SPARQL. As RIF Assembler aims to keep things simple, it uses the simpler equivalence of annotations, as suggested by the RIF-in-RDF note.

The mapping $\pi$ can be used to comprehensively collect all the annotations of multiple RIF documents as a single RDF graph. This makes it possible to execute SPARQL queries against the rules metadata, as well as fostering information reuse based on "linked data" principles. Moreover the graph structure of RDF is a more natural structure than XML trees to represent relationships between rules (e.g., rule A replaces rule B) and rules belonging to multiple rulesets.

**Table 1.** Interpretation of RIF annotations as RDF graphs (N3 syntax)

| Annotation $\varphi$ | $\pi(\varphi)$ |
|---|---|
| `s [p -> o]` | `{ s p o }` |
| `s[`$\mathtt{p}_1$`->`$\mathtt{o}_1$ ... $\mathtt{p}_n$`->`$\mathtt{o}_n$`]` | `{s` $\mathtt{p}_1$ $\mathtt{o}_1$ `;` $\cdots$ `;` $\mathtt{p}_n$ $\mathtt{o}_n$ `}` |
| `And(`$\mathtt{F}_1$`, ... ,` $\mathtt{F}_n$`)` | $\{\pi(\mathtt{F}_1)\} \cup \cdots \cup \{\pi(\mathtt{F}_n)\}$ . |

**Table 2.** RDF interpretation of RIF hierarchies

| RIF PS non-terminal symbol | $\pi(\cdot)$ |
|---|---|
| ⟨ `Group` ⟩ | { `<group`$_{id}$`> rdf:type rulz:Ruleset` } |
| ⟨ `RULE` ⟩ | { `<rule`$_{id}$`> rdf:type rulz:Rule` } |
| ⟨ `Group'` ⟩ within ⟨ `Group` ⟩ | { `<group`$_{id}$`> rulz:subset <group'`$_{id}$`>` } |
| ⟨ `RULE` ⟩ within ⟨ `Group` ⟩ | { `<rule`$_{id}$`> rulz:inRuleset <group`$_{id}$`>` } |

In order to represent the structure of RIF documents in RDF graphs, we make use of the Rulz vocabulary[2] to type the resources (rules and groups) and to capture their hierarchy. The mapping between RIF entities and Rulz concepts is described in Table 2. The expression $< \cdot_{id} >$ denotes the URI that identifies the RIF entity (that is, the `id`). If the entity has no identifier, a fresh blank node is generated and associated with the entity.

### 2.2 From RDF Graphs to RIF Documents

To translate from RDF back to RIF, the $\pi^{-1}$ mapping is applied. The hierarchy described by some RDF graphs cannot be mapped by $\pi^{-1}$ because the data structures are not isomorphic — a graph (RDF) is more general than a tree (XML). To ensure the transformation is feasible, a constraint is introduced: each of the connected subgraphs defined by the edges labelled `rulz:subset` and `rulz:inRuleset`$^{-1}$ must have a tree-shaped structure. In other words: a ruleset can have no more than one super-ruleset, no cycles can occur within a subset hierarchy and a rule can be part of no more than one ruleset.

The $\pi^{-1}$ mapping may produce a single tree or many of them. If there are multiple trees, i.e, there are disconnected rulesets and rules, a new root node ⟨`Group`⟩ is generated to subsume all the structures under a single tree. Moreover, if the output of the mapping is a single rule that is not part of any group, a root node ⟨`Group`⟩ is also generated, as required by the RIF/XML syntax.

As indicated at the beginning of this section, RIF does not permit metadata to be attached to an entity that lacks an identifier. Note that this restriction

[2] `http://vocab.deri.ie/rulz`

does not exist in RDF because blank nodes are allowed as the subject of triples. Consequently, if blank nodes are present in the RDF graph, the $\pi^{-1}$ mapping mints identifiers (URIs) for rules and rulesets to complete the transformation.

## 3 RIF Assembler

RIF Assembler receives one or more RIF documents as input and produces a single RIF file. The instructions to select the domain rules that will form part of the output and drive their transformation are also provided by rules. These assembly instructions are metarules (i.e., second-order rules). The tool can also read OWL ontologies and RDF datasets, which are taken into account in the metarule reasoning process, but are not part of the output of the system.

RIF Assembler does not create rules from scratch. Any rule in the output must be present in the input (note that the reverse is not necessarily true), although it may be subject to changes during the process. The transformations are limited to its metadata and its location within the structure of the ruleset. More specifically, it is not possible to modify the formulation (and therefore the meaning) of the rule.

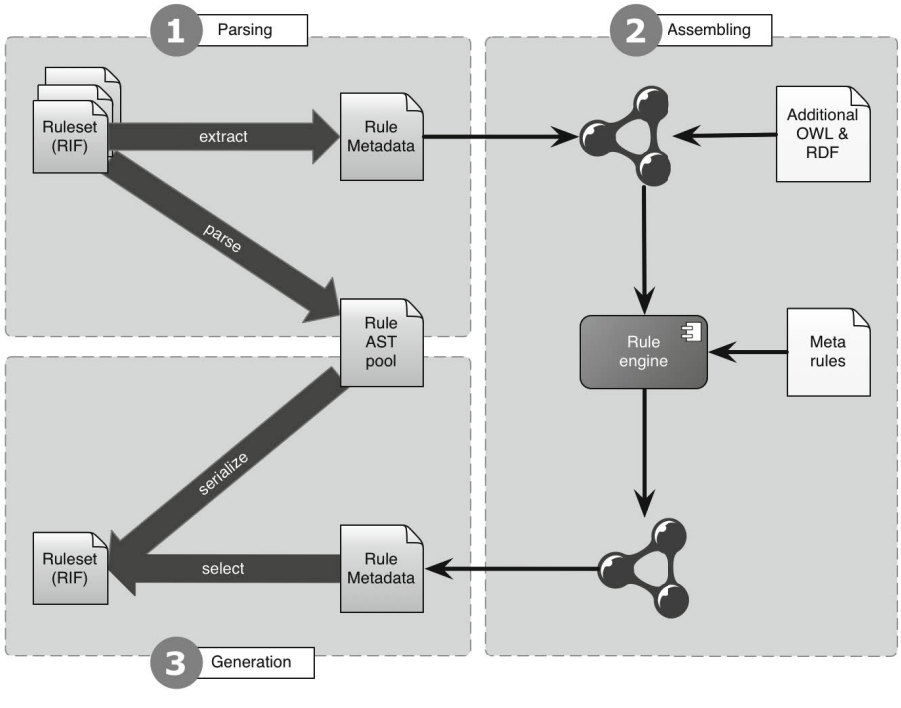


**Fig. 2.** Architecture of RIF Assembler

RIF Assembler carries out the following steps, as illustrated in Figure 2:

**RIF parsing:** The input RIF documents are parsed to populate two data structures[3]. Individual Abstract Syntax Trees (AST) are generated for each rule, and incorporated into a pool. As a consequence, rules are disengaged from the source documents. At the same time, the rule and group metadata, as well as the hierarchical organization of the original documents, are converted to RDF as explained in the previous section. This information is put into an RDF store and merged with other domain knowledge sources, such as OWL ontologies and RDF datasets.

**Assembling:** In the previous step rules and groups are abstracted from their sources and syntax. This makes it possible to handle them as RDF resources and to manipulate them by simply querying and updating the RDF graph. Metarules specify the conditions and restrictions to be satisfied in order to produce the tailored system. More precisely, metarules are fired to create, modify and delete rule and group metadata; to select and delete rules; or to rearrange the hierarchy by creating/deleting groups and changing rule membership. It is worth noting that domain rules and metarules live in separate universes; at no point do they mix with each other.

**RIF generation:** The last step of RIF Assembler generates a single AST from the individual ASTs available in the rule pool. The RDF graph is queried to find out which rules and groups are to be included in the output and how they nest. The annotations are also obtained from the RDF graph. The unified AST is then serialized as a RIF/XML document.

This process has been implemented in a web application built upon the Jena Framework[4]. To implement the metarules, the forward chaining engine of the Jena general purpose rule-based reasoner (also known as Jena Rules) is used[5]. Input and output RIF documents use the RIF/XML syntax instead.

A live instance of the application is available at `http://ontorule-project.eu/rifle-web-assembler/`. The user interface is shown in Figure 3 and is divided in three areas:

1. A toolbar to execute the main operations, namely: (a) to export the graph to an RDF/XML file; (b) to execute SPARQL queries; (c) to export the assembled rules to a RIF document; (d) to compute a partial evaluation of the assembly and (e) to reset the application. The partial evaluation is a feature that supports multistage assembly processes.
2. A list of the input documents including: domain rules in RIF/XML, domain knowledge in RDF and OWL files, and metarules in Jena rules. The documents can be uploaded from local files, URIs or by direct input (typing or pasting the text in a form).

[3] To parse RIF/XML, we use the RIFle library, available at `http://rifle.sf.net`.
[4] `http://incubator.apache.org/jena/`
[5] `http://incubator.apache.org/jena/documentation/inference/`


**Fig. 3.** User interface of RIF Assembler tool

3. The panel at the right displays statistics about the information loaded in the system and the resulting model after applying the metarules. For instance, in Figure 3, 4 domain rules have been loaded from one RIF document, but only 3 rules remain after the execution of the metarules.

Finally, RIF Assembler makes use of Parrot, a RIF and OWL documentation tool [16], in order to display the combinations of ontologies and rules in the input files and the final assembled ruleset.

## 4 Usage Scenarios

Two usage scenarios are presented in this section. The first one is a simplistic, imaginary example in the health care domain, and illustrates the concept of rule mashups. The second one is a realistic and more sophisticated scenario related to knowledge reuse within ArcelorMittal, the world's largest steel producer. More details about the latter scenario can be found at [7].

### 4.1 Health Care Scenario

Nowadays, drugs are shipped with Patient Information Leaflets (PIL) which contain essential information about the product. The structure of a PIL, e.g., "list of excipients", "contra-indications", and "use during pregnancy and lactation", is defined by regulations, for instance, European Directive 2001/83/EC.

One can imagine that, in the near future, pharmaceutical companies will publish this information on the web (open data). Although some information, such as

the list of excipients, may be modeled and made available as RDF graphs using vocabularies such as SNOMED[6], other information, such as contra-indications and interactions with other medicinal products, may require using RIF rules. As an example, Figure 1 contains a rule that alerts breast-feeding women not to take a particular drug.

In this hypothetical scenario, RIF Assembler may be used to process the rules harvested from the web. As not all the sources are equally trustworthy, RIF Assembler may execute metarules that decide which rules are reliable, taking into account, for instance, provenance information contained in annotations. An example of these metarules would be: "keep all the rules that come from a source whitelisted by the World Health Organization". The output of the assembly process would be a ruleset potentially usable for making decisions on treatments and prescriptions. Such a system would support professionals in medicine and be the foundation for personal software assistants (virtual doctors).

### 4.2 Steel Industry Scenario

This real-world scenario focuses on the quality control system of a galvanization line of the ArcelorMittal steel mill. After a steel coil has been galvanized, the product quality is assessed by examining data measured by sensors all along the production line. Three alternative outcomes are possible: if the coil meets the quality requirements from the order, it is shipped to the customer; if the coil presents some non-critical defects, it is sent to repair; finally, if the product quality is critically low, it is sent for scrapping.

As one of the world's leaders in the steel industry, ArcelorMittal operates several factories all over the globe. Ideally, the rules that implement company business policies should be usable wherever a factory carries out the galvanization process. Actually, the situation is more complex; factories are not identical clones. Steel mills are equipped with diverse machinery in terms of precision, age, maintenance, operation-cost and lifespan. Moreover, although rule-based systems are widely adopted within the company, different rule engines and rulesets are used at each location, for historic reasons as well as local specifics. The company maintains a pool of shared knowledge that is populated by the artifacts already deployed within the company. Thanks to RIF and RIF Assembler, rules can be drawn from this pool and packaged into rulesets tailored for a given factory. More precisely, RIF Assembler, driven by the metarules, builds a system that assesses the quality of galvanized coils for a particular facility, namely, the factory located in Avilés, Spain.

In a preliminary step, rules in the pool were annotated with provenance information (e.g., the factory they come from) and regarding the part of the business process they carry out (or *realize*, in the vocabulary of this scenario). To this end, we have created a domain ontology that models industrial processes, such as galvanization, and their associated quality processes. In essence, a business process is split into a sequence of tasks. The `implements` property captures the

[6] `http://www.snomed.org/`

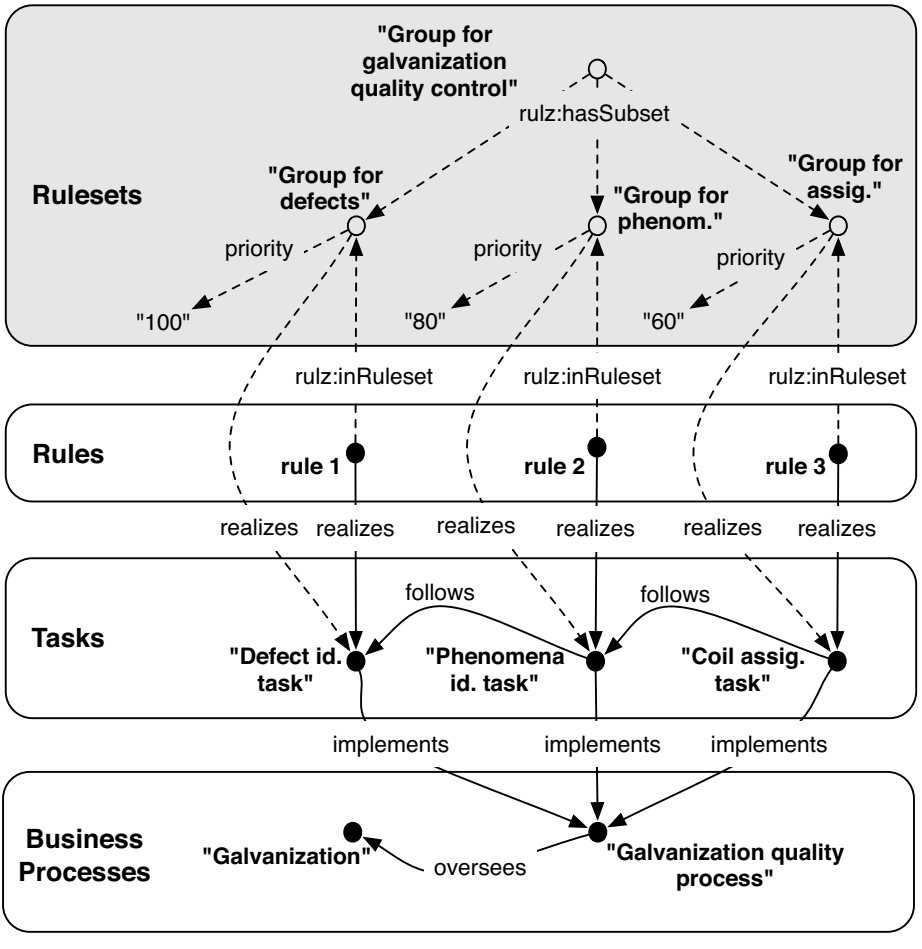


**Fig. 4.** RDF graph representing the rules and rulesets of the steel industry scenario

relation between tasks and business processes. The `follows` property defines total order within sets of tasks. The `realizes` property is used in rules and ruleset annotations to connect them to the tasks (see Figure 4). Using this ontology, we can express the fact that "The group to which Rule 1 belongs realizes the defect identification task that implements the galvanization quality process".

Rules are drawn from the shared pool and their metadata are augmented and refined with specific knowledge borrowed from other ArcelorMittal facilities. Metarules instruct RIF Assembler on how to manipulate and organize the input rules. These assembly instructions are dictated by business experts in the domain who are aware of the specifics throughout the production line. In the case of the Avilés galvanization line, the metarules carry out three activities:

1. Creating a generic quality system from the rule pool. Metarules such as the one in Figure 5 select the rules that are relevant to the galvanization quality

process and put them in a group as depicted in Figure 4. Rules from the pool that are not relevant are simply discarded. Another metarule assigns priorities to each group based on the precedence relation between tasks.
2. Augmenting the system with additional rules from similar factories. Some metarules determine which business policies in use in other facilities may also be applied in Avilés, even if the associated rules are not part of the generic quality system. These rules are simply added to the final ruleset.
3. Refining the system by attending to the specifics of the galvanization process in Avilés. In some cases, rules from another factory may also be added, replacing rules from the generic quality system. This is the case with rules related to electrogalvanization, which is a refinement of the generic galvanization process and is available only at certain factories.

Although not used in this scenario, RIF Assembler may also exploit OWL inference, particularly the OWL-RL profile [12], to augment the expressivity of metarules. This opens the door to handling ontologies with complex hierarchies, such as those that describe business processes beyond the simple case addressed in this scenario. Another practical application of reasoning within RIF Assembler is to combine rules that are annotated with respect to different ontologies, and that consequently require alignment.

Figure 4 depicts the RDF graph at an intermediate stage of the assembly process. The solid lines represent annotations of rules and statements from the ontology that is provided as input. The dashed lines indicate inferences derived by the metarules. All the ruleset resources and their hierarchy were created by the metarules.

```
[R1:
    (?rule rdf:type rulz:Rule)
    (?rule bp:realizes ?task)
    (?rule rulz:inRuleset ?group)
    (?rule bp:factory bp:Pool)
    (?task bp:implements bp:QualityGalvanizationProcess)
    (?task rdfs:label ?taskName)
    strConcat("Autogenerated ruleset for task ",?taskName,?rulesetName)
    makeTemp(?newGroup)
->
    remove(2)
    (?newGroup rdf:type rulz:Ruleset)
    (?newGroup rdfs:label ?rulesetName)
    (?newGroup bp:realizes ?task)
    (?newGroup bp:scope bp:GalvanizationQualityProcess)
    (?newGroup bp:factory bp:FactoryInAviles)
    (?rule rulz:inRuleset ?newGroup)
]
```

**Fig. 5.** Metarule that creates new groups for rules from the pool that realize a relevant business task

## 5 Related Work

Business Rule Management Systems (BRMS) evolve from (production) rule engines to cope with other requirements beyond the execution of rules. One of the key features of modern BRMSs is the rule repository [9], where artifacts and their metadata are stored. Rule metadata are particularly important in the last step of the BRMS lifecycle: maintenance of the rule-based application [13]. This final step involves knowledge reuse and adaptation to changes in the application context. Leading BRMS products feature some mechanism to extract rules, i.e., generate rulesets that contain subsets of the complete knowledge base, by specifying conditions applied to rule metadata [3]. This is the case of IBM WebSphere ILOG JRules, which permits the execution of queries against rule metadata. Improving on this feature was one of the motivations for our work on knowledge reusability. In particular, RIF Assembler extends the query functionality with the execution of metarules, permitting not only selection, but also modification of the ruleset structure.

The Object Management Group[7](OMG) introduced the Model-driven Architecture (MDA) paradigm [15], which aims to model real systems using standards such as UML, MOF or XMI. The models defined with these standards remain abstract, and actual implementations can be obtained via automatic code generation. Although the topic of knowledge reuse is shared with RIF Assembler, different drivers motivate these approaches: MDA is model-centric while RIF Assembler is rule-centric.

SPIN [11] is a W3C Member Submission that proposes an RDF syntax for SPARQL queries. Some of these queries, namely CONSTRUCT and SPARUL queries, may express rules [14]. Therefore this work and SPIN share the idea that rules can be represented by RDF resources. This permits the construction of hybrid models that combine the model (ontology) and the queries, and where queries can modify the model itself. We chose to build on RIF and not on SPARQL/SPIN, because the former covers a wider range of rule languages. Thus, RIF Assembler can be used with any BRMS that supports the RIF standard, while using SPIN would require translation from different rule languages to SPIN.

XSPARQL provides a language to transform between RDF and XML [1]. It is conceivable to use it to generate RIF/XML from SPARQL queries. In this sense, it can go beyond what it is currently possible with RIF Assembler. However there is a price to pay for this flexibility: while RIF Assembler only requires rule writing skills, which is an ability that is presumed in the target user community, XSPARQL requires technical knowledge of the RIF/XML syntax, the RDF model and the SPARQL query language.

There are similarities between our work and metaprogramming, i.e., programs that generate other programs [17]. RIF Assembler can be seen as metaprogramming where both the final program and the metaprogram are expressed in rules (RIF and Jena Rules, respectively). However, RIF Assembler does not alter the formulation of the rules, and therefore it is limited with respect to the

[7] `http://www.omg.org/`

expressivity of the output programs (rulesets). In the future, we plan to fully translate the rules to RDF in order to enable manipulation of the rule syntactic structure. Moreover, at this stage we do not fully exploit the ability of rules to check the consistency of the output.

## 6 Conclusions

The vision of RIF Assembler relies on an appealing idea, namely, that rules can be reified as RDF resources and treated as first-class web citizens. By doing so, the doors are opened to rules linking to arbitrary resources in the web of data, and vice versa. Many applications may exploit this idea, such as rule search engines and rule-based personal assistants in the areas of ambient intelligence and eHealth.

In this paper, two scenarios give insight into the potential of RIF Assembler. We show that, assuming that annotated rules are available, it is possible to derive mashup rulesets by simply writing down assembly instructions as metarules. These metarules can be inspired by domain experts, dramatically simplifying the construction of families of decision-support systems with respect to previous, manual approaches.

Knowledge reuse, in particular the reuse of rules, is of critical importance to any organization. We envision that the functionality of RIF Assembler may eventually be an integral part of future BRMS products. As the availability of rules increases on the web and in corporate environments, fostered by the adoption of RIF, reuse will become easier. However, RIF Assembler goes beyond the pure exchange of rules. It proposes that rules can be mixed and manipulated independently of their source. For instance, given two rule-based systems $A$ and $B$, respectively developed with IBM JRules and JBoss Drools (both Java-based environments), their rules can be exported to RIF. This permits the use of RIF Assembler to select subsets of $A$ and $B$ to create a new system (described in RIF), $C$, that can be translated to JRules, Drools or any other execution environment, such as the C-based CLIPS. RIF Assembler supports scenarios where reuse implies more than the mere portability of previously-built solutions, but also rearrangement of knowledge in order to meet new requirements and contexts of use, as in the ArcelorMittal scenario.

RIF Assembler does not yet analyze the contents and semantics of the rules. Therefore, it is difficult to detect and handle contradictions between the rules. Similarly, RIF Assembler does not provide automated consistency checks of the assembled ruleset. It is up to the user to decide and implement metarules to deal with rulesets that do not merge seamlessly. Nevertheless, the tool provides some help in this task. For instance, it makes it easy to replace a troublesome rule with a better alternative. The authors will further improve in this direction following the findings made by the ONTORULE project on static rule consistency checking [6].

**Acknowledgements.** This work has been partially supported by the European Commission under ONTORULE Project (FP7-ICT-2008-3, project reference 231875).

## References


1. Akhtar, W., Kopecký, J., Krennwallner, T., Polleres, A.: XSPARQL: Traveling between the XML and RDF Worlds – and Avoiding the XSLT Pilgrimage. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 432–447. Springer, Heidelberg (2008)
2. Boley, H., Kifer, M.: RIF overview. W3C Working Group Note, W3C (June 2010), http://www.w3.org/TR/rif-overview/
3. Boyer, J., Mili, H.: Agile Business Rule Development - Process, Architecture, and JRules Examples. Springer (2011)
4. de Bruijn, J.: RIF RDF and OWL Compatibility. Recommendation, W3C (June 2010), http://www.w3.org/TR/rif-rdf-owl/
5. de Sainte Marie, C., Hallmark, G., Paschke, A.: RIF Production Rule Dialect. Recommendation, W3C (June 2010), http://www.w3.org/TR/rif-prd/
6. Fink, M.: D2.6 consistency maintenance final report. Deliverable, ONTORULE project (2011)
7. González-Moriyón, G., Polo, L., Berrueta, D., Tejo-Alonso, C.: D5.5 final steel industry public demonstrators. Deliverable, ONTORULE project (2011)
8. Hawke, S., Polleres, A.: RIF In RDF. Working Group Note, W3C (May 2011)
9. Herbst, H., Myrach, T.: A repository system for business rules. In: Mark (ed.) Database Application Semantics, pp. 119–138. Chapman & Hall, London (1997)
10. Kifer, M., Boley, H.: RIF Basic Logic Dialect. Recommendation, W3C (June 2010), http://www.w3.org/TR/rif-bld/
11. Knublauch, H., Hendler, J.A., Idehen, K.: SPIN - Overview and Motivation. Member Submission, W3C (February 2011)
12. Motik, B., Fokoue, A., Horrocks, I., Wu, Z., Lutz, C., Grau, B.C.: OWL 2 web ontology language profiles. W3C recommendation, W3C (October 2009), http://www.w3.org/TR/2009/REC-owl2-profiles-20091027/
13. Nelson, M., Rariden, R., Sen, R.: A lifecycle approach towards business rules management. In: Proceedings of the 41st Annual Hawaii International Conference on System Sciences, pp. 113–113 (January 2008)
14. Polleres, A.: From sparql to rules (and back). In: Proceedings of the 16th International Conference on World Wide Web WWW 2007, p. 787 (2007)
15. Poole, J.D.: OMG Model-Driven Architecture Home Page (2001), http://www.omg.org/mda/index.htm
16. Tejo-Alonso, C., Berrueta, D., Polo, L., Fernández, S.: Metadata for Web Ontologies and Rules: Current Practices and Perspectives. In: García-Barriocanal, E., Cebeci, Z., Okur, M.C., Öztürk, A. (eds.) MTSR 2011. CCIS, vol. 240, pp. 56–67. Springer, Heidelberg (2011)
17. Visser, E.: Meta-programming with Concrete Object Syntax. In: Batory, D., Consel, C., Taha, W. (eds.) GPCE 2002. LNCS, vol. 2487, pp. 299–315. Springer, Heidelberg (2002)