Rule Interchange Format: The Framework

Michael Kifer

State University of New York at Stony Brook, USA

Abstract. The Rule Interchange Format (RIF) is an activity within the World Wide Web Consortium aimed at developing a Web standard for exchanging rules. The need for rule-based information processing on the Semantic Web has been felt ever since RDF was introduced in the late 90's. As ontology development picked up pace this decade and as the limitations of OWL became apparent, rules were firmly put back on the agenda. RIF is therefore a major opportunity for the introduction of rule based technologies into the main stream of knowledge representation and information processing on the Web.

Despite its humble name, RIF is not just a format and is not primarily about syntax. It is an extensible framework for rule-based languages, called RIF *dialects*, which includes precise and formal specification of the syntax, semantics, and XML serialization. In this paper we will discuss the main principles behind RIF, introduce the RIF extensibility framework, and outline the Basic Logic Dialect—the only fully developed RIF dialect so far.

1 Introduction

The *Rule Interchange Format* (RIF) activity within the World Wide Web Consortium (W3C) aims to develop a standard for exchanging rules among disparate systems, especially on the Semantic Web. Given that the existing rule systems, both commercial and research prototypes, have wide variety of features and differ not only syntactically but also—more importantly—semantically, the goal of the RIF effort is not at all simple. Some systems extend one another syntactically and/or semantically, but in many cases this is true only to a degree. Other rule systems are largely incompatible, each having features that the other system does not. With this diversity, how can interoperability be achieved?

The vision of RIF is a collection of *dialects*—an extensible set of languages with rigorously defined syntax and semantics. Extensibility here means that new dialects can be added, if sufficient interest exists, and the languages are supposed to share much of the syntactic and semantic apparatus.

Because of the emphasis on rigor and semantics, the term "format" in the name of RIF might seem a misnomer. However, making a cute acronym is not the only reason for the choice of this term. The idea behind rule exchange through RIF is that the different systems will be able to map their languages (or substantial parts thereof) to and from the appropriate RIF dialects in *semantics-preserving* ways and thus rule sets and data could be communicated by one

system to another provided that the systems can find a suitable dialect, which they both support. The intermediate RIF language is supposed to be in the XML format, whence the term "format" in the RIF name.

The RIF Working Group has plans to develop two kinds of dialects: logic-based dialects and dialects for rules with actions. The logic-based dialects include languages based on first-order logic and also a variety of logic programming dialects based on the different non-first-order semantics such as the well-founded and stable semantics [22, 12]. The rules-with-actions dialects will be designed for production rule systems, such as Jess and Drools [15, 14], and for reactive rules such as those represented by XChange [4], FLORA-2 [16], and Prova [18]. At the time of this writing, only the *Basic Logic Dialect*, RIF-BLD (which belongs to the first category), has been substantially completed and is in the "Last Call" status in the W3C standardization process [2]. In the second category, a *Production Rule Dialect*, RIF-PRD, is under active development [8].

These plans are very ambitious, and in the beginning it was not at all obvious how the different dialects could be made to substantially share syntactic and, especially, semantic machinery. Even within the logic-based category the dialects are expected to have vastly different semantics: the first-order semantics warrants inferences that are different from those warranted by the logic programming semantics, and the various logic programming semantics do not agree in all cases. This is where the RIF extensibility framework comes in. At present, only the *Framework for Logic Dialects*, RIF-FLD, has been worked out to sufficient degree of detail [3], and this is the main subject of this paper.

We assume general familiarity with first-order logic syntax and semantics, and with the idea of rule-based languages, especially logic programming languages like Prolog [7]. We also assume that the reader understands the difference between first-order logic based languages and those based on logic programming; especially the difference that the notion of *negation* plays in both.

This survey is organized as follows. In Section 2, we give an overview of the RIF framework for logic dialects. Section 3 describes the syntactic machinery of the framework, including the notions of terms, formulas, signatures, and well-formedness. Section 4 describes the semantic framework. In Section 5 we give a brief introduction to the Basic Logic Dialect of RIF and show how it can be described in terms of the RIF framework. Section 6 concludes the paper.

2 RIF Framework for Logic Dialects—An Overview

The RIF *Framework for Logic Dialects*, RIF-FLD, is a formalism for specifying all logic dialects of RIF, including the Basic Logic Dialect [1]. In itself, FLD is a logic in which both the syntax and semantics of the dialects are described through a number of mechanisms that are commonly used in practice and in literature, but are rarely brought all together. Fusion of all these mechanisms is required because the framework must be broad enough to accommodate several different types of logic languages and because various advanced mechanisms are needed to facilitate translation into a common framework. RIF-FLD gives precise

definitions to these mechanisms, but allows details to vary. The design of RIF envisages that its future logic dialects will be based on RIF-FLD and will be defined as specializations of FLD. Being all derived from the same framework will ensure that RIF dialects are syntactically and semantically compatible in the sense that extensions, restrictions, and common subsets of the different dialects will be formally identifiable and rule systems would be able to communicate their rule sets using a collection of such dialects.

The framework has three main components: the syntactic framework, the semantic framework, and the XML framework. The syntactic framework defines a general mechanism for specifying which kinds of terms and formulas are allowed and how to specialize this mechanism to produce specific dialects. The semantic framework provides model-theoretic mechanisms for specifying how logical inference is to be defined in the derived dialects. The XML framework defines the general principles for mapping the syntax of RIF-FLD to a concrete XML interchange format.

As an example of this approach, the RIF Basic Logic Dialect is normatively defined as a specialization of RIF-FLD. Having RIF-FLD is a major advantage because the specification of RIF-BLD as a specialization of RIF-FLD is very short and easy to grasp. For comparison, RIF-BLD is also specified directly, without relying on the framework. This specialization is also normative, but much longer and more complex. It is required that the two specifications of BLD are equivalent and any discrepancy must be treated as a mistake to be corrected.

In the following sections, we will provide an informal survey of the syntactic and semantic frameworks. It is informal both in order to be brief and also because the reader is encouraged to consult the definitive document [3].

3 The Syntactic Framework

The syntactic framework defines the types of terms and formulas that are allowed in a dialect. A specific dialect might choose to restrict certain combinations of symbols and throw out some combinations altogether.

3.1 Terms: The Object Level

The framework defines the following types of terms (among others: it is not the purpose this this survey to complete):

- *Constants and variables.* In the RIF presentation syntax, variables are denoted using alphanumeric symbols prefixed with a "?"-mark, and we will also do so here.
- *Positional terms.* If t and t_1, ..., t_n are terms then $t(t_1 \ ... \ t_n)$ is a positional term.[1] These are like the usual terms of first-order logic except that

[1] The presentation syntax of RIF does not use commas to separate the arguments in predicates and terms. It is an abstract syntax and, as such, it omits certain details that might be important for unambiguous parsing.

the symbols are not necessarily partitioned into individuals, functions, and predicates (any such restrictions are left to the dialects' discretion). In addition, variables are allowed to occur anywhere a term can. Thus, a positional term can be as general as a HiLog term [5] and expressions of the form $?X(abc\ ?W)(?Y\ ?Z(?V\ 33))$ are well within the limits of what is allowed.

- *Terms with named arguments.* The arguments of a term can be named as in $person(name{\rightarrow}Bob\ age{\rightarrow}33)$. Such a term is distinct from, say, $person(Bob\ 33)$ or $person(spouse{\rightarrow}Bob\ age{\rightarrow}33)$. However, the *order* of the named arguments within such a term is immaterial, so $person(name{\rightarrow}Bob\ age{\rightarrow}33)$ and $person(age{\rightarrow}33\ name{\rightarrow}Bob)$ are indistinguishable.
- *Frame and classification terms.* RIF-FLD includes certain terms borrowed from F-logic [17]. A *frame* term has the form $t[p_1{\rightarrow}v_1\ ...\ p_n{\rightarrow}v_n]$, where t, p_1, ..., p_n, v_1, ..., v_n are terms. The order of the attribute specifications (the $p_i{\rightarrow}v_i$'s) is immaterial, like in the case of terms with named arguments. However, frames have very different semantics compared to the named argument terms. For instance, in $bob[name{\rightarrow}Bob\ age{\rightarrow}33]$, *bob* denotes an object and $name{\rightarrow}Bob$, $age{\rightarrow}33$ are statements about the properties of that object. In contrast, $person(name{\rightarrow}Bob\ age{\rightarrow}33)$ is not a statement about the object *person*. Here *person* is the name of a relation type (think of a database table) and $name{\rightarrow}Bob\ age{\rightarrow}33$ describes a particular relation of that type. So, $bob[spouse{\rightarrow}mary]$ would still be a statement about the same object *bob* (just about some other of its properties), while the statement $person(spouse{\rightarrow}Mary)$ would have no relationship to the earlier statement $person(name{\rightarrow}Bob\ age{\rightarrow}33)$.
  Classification terms include *membership* and *subclass* terms. Here $t\#s$ represents a membership relationship between the member-object t and the class-object s; $s\#\#c$ is a term that represents the subclass relationship between the objects s and c.[2] For instance, $student\#\#person$.
- Other kinds of terms include equality and external terms. The latter represent references to outside sources of information and built-ins.

Since communication between the different rule systems through the medium of RIF is supposed to be by translation, one might ask why so many different kinds of terms? After all, it is well known that everything can be encoded using just the first-order terms; in fact lists alone suffice. The answer is *model-preservation* or *round-tripping.* One of the requirements in RIF is to support round-tripping, i.e., the ability to translate a rule set from, say, system S_1 to RIF, then to S_2, then back to RIF, back to S_1, and get not only a semantically equivalent set of rules, but essentially the same set of rules from the modeling points of view. What this is supposed to mean precisely has not been addressed, but the intuitive idea is that if something was modeled as an object (a frame term) then it should stay an object and not metamorphose itself into a relation (a positional or a named argument term) after returning back to S_1. Likewise,

[2] Those familiar with F-logic might be surprised to see $t\#s$ and $s\#\#c$ instead of $t : s$ and $s :: c$, but the colon has been irrevocably appropriated for other purposes in the world of W3C standards.

the subclass and membership relationships are well-established modeling primitives and must be recognized by the syntax. This also simplifies translation to and from RIF, and makes it more natural.

The other question that comes to mind is why things that are normally called formulas (e.g., frames and classification terms in F-logic [17]) are called *terms* in RIF-FLD? The answer is that RIF-FLD is required to support a degree of *reification*—the ability to represent formulas (which are statements about true facts or beliefs) as terms (i.e., objects). In this way, RIF will allow dialects in which statements can be made about other statements, and these *meta*-statements can then be processed by rules.

3.2 Formulas: The Statement Level

The logic RIF framework defines several types of formulas, most of which are adaptations from other known logics. However, in RIF-FLD they are all together in one logic system.

- *Atomic formulas*: A term is also an atomic formula. Like in HiLog [5], this blurs the distinction between objects and statements about objects and lays a foundation of the infrastructure for meta-reasoning in RIF dialects that might choose to support it.
- *Conjunction and disjunction*: These are the usual connectives in first-order logic. The RIF syntax for that is `And`$(\phi_1 \ldots \phi_n)$ and `Or`$(\phi_1 \ldots \phi_n)$.
- *Negation*: RIF-FLD supplies both the classical negation as used in first-order logic, denoted `Neg`, and a symbol for default negation, as used in logic programming. The latter is intended for logical notions of default negation, such as those based on the well-founded and the stable-model semantics [22, 12]—*not* for negation-as-failure, as used in Prolog [6]. In view of this, the current choice of the symbol for default negation, `Naf`, is misleading and might be replaced in the future. It is also possible that *explicit negation* (a weaker form of classical negation that is sometimes used in logic programming [13]) might be added in the future.[3]
- *Rule implication*: A rule implication is a formula of the form *phi* `:-` ψ. This is the notion of implication as used in logic programming; it is different from the classical implication and is *not* equivalent to `Or`$(\phi$ `Neg` $\psi)$.
- *Quantification*: A quantified formula is, as usual, a formula of the form `Forall` $?V_1 \ldots ?V_n\,(\phi)$ or `Exists` $?V_1 \ldots ?V_n\,(\phi)$.

Apart from these, FLD also has `Group`-formulas and `Document`-formulas. A group formula is simply a set of formulas of the above form, and groups can be nested. This type of formulas exists just for convenience and for possible future enhancements. One convenience is the ability to assign an identifier (say, a URL) and meta-data to a group of formulas. This information can then be used in other Web documents.

[3] Since true classical negation and explicit negation are never used together, it is also possible that `Neg` will be used for both.

A `Document`-formula generalizes what we earlier informally called a "rule set." The Web consists of documents and this is also a structural unit chosen for RIF. An important aspect of documents is that one can *import* the other. This provides a degree of modularity similar to what exists in other Web standards, such as XML Schema and OWL [11, 9].

Documents also provide a convenient way to localize constant symbols to particular documents and avoid clashes. This is particularly important for logic programming languages where it is common to use intermediate predicates that are not supposed to have meaning outside of a particular document.

3.3 Signatures: The Key to Extensibility

One of the most important ingredients that makes RIF-FLD into a *framework* for defining other languages (dialects) is the concept of a signature. Signatures determine which terms and formulas are *well-formed*. It is a generalization of the notion of a *sort* in classical first-order logic [10]. Each symbol has an associated signature. A signature defines, in a precise way, the syntactic contexts in which the symbol is allowed to occur.

For instance, the signature associated with a symbol p might allow p to appear in a term of the form *f(p)*, but disallow it to occur in the term *p(a,b)*. The signature for *f*, on the other hand, might allow that symbol to appear in *f(p)* and *f(p,q)*, but disallow *f(p,q,r)* and *f(f)*. Note that, say, *f(f)* is still a term according to our earlier definition; it is just not a *well-formed* term. In this way, it is possible to control which symbols are used for predicates and which for functions, where variables are allowed to occur and where they are not allowed.

A *signature* is a statement of the form $\eta\{e_1,\ ...,\ e_n,\ ...\}$ where η is the name of the signature and $\{e_1,\ ...,\ e_n,\ ...\}$ is a countable set of *arrow expressions*. The number of such expressions in a particular signature can be zero or more, or it can be infinite. The dialects decide for themselves. In RIF-BLD, signatures can have at most one arrow expression. Dialects that support polymorphism may allow more than one arrow expression in a signature. HiLog [5], for example, puts a countably infinite number of arrow expressions in all signatures.

An *arrow expression* is a statement of the form $(\kappa_1\ ...\ \kappa_n) \Rightarrow \kappa$, where κ, κ_1, ..., κ_n are signature names. For instance, if *term* is a signature name then $(\,) \Rightarrow term$ and $(term) \Rightarrow term$ are signatures.

There is more to the notion of arrow expression that the above suggests. For instance, the above are arrow expressions for just the *positional* terms. There are also signatures for terms with named arguments, frames, and signatures can be organized into class hierarchies. However, we will ignore these aspects and focus on the essentials.

Signatures are used to control the context in which symbols occur using the notion of *well-formedness*. Earlier we defined the notion of terms and formulas, but those definitions do not say whether a term or a formula is well-formed. In order to define this notion we must assume that every symbol in the alphabet of the language is assigned a unique signature. How exactly this is done depends on a dialect. For instance, BLD imposes very strict conditions on signatures, which

makes it possible to assign signatures by the context in which the symbols are used. Terms are well-formed if their structure conforms to the following rules.

- A constant or variable symbol with signature η is a well-formed term with signature η.
- A term $t(t_1 \ ... \ t_n)$ is well-formed and has a signature σ if and only if
  - `t` is a well-formed term that has a signature that contains an arrow expression of the form $(\sigma_1 \ ... \ \sigma_n) \Rightarrow \sigma$; and
  - Each t_i is a well-formed term with signature σ_i.

This is not a full definition. It omits terms with named arguments, frames, membership and subclass terms, and other aspects. The full definition can be found in [3]. However, this partial definition should convey the idea. For instance, if p has the signature $mysig\{(obj) \Rightarrow obj, (obj \ obj) \Rightarrow obj, (obj \ obj \ obj) \Rightarrow obj\}$ and a, b, c each has the signature $obj\{\}$ then $p(p(a) \ p(a \ b \ c))$ is a well-formed term with signature $obj\{\}$. On the other hand, $p(a \ b \ c \ a)$ is a term, but not a well-formed one, since the signature of p has no arrow expression that permits p to have four arguments. The following is an even more telling example. Suppose *John* and *Mary* are symbols with the signature $obj\{\}$, the variable *?P* has the signature $h_2\{(obj \ obj) \Rightarrow obj\}$, and *closure* has the signature $h_3\{(h_2) \Rightarrow p_2\}$, where p_2 is the name of the signature $p_2\{(obj \ obj) \Rightarrow obj\}$. Then $?P(John \ Mary)$ and $closure(?P)(John \ Mary)$ are well-formed terms with signature $obj\{\}$.

Designers of each particular RIF dialect can decide which signatures can be assigned to which symbols and in this way fully determine the syntax of the dialect. Thus, RIF-FLD provides a general framework, which dialects can use to specify their syntaxes. The present draft of RIF-BLD uses a different technique for defining well-formed formulas, but a future draft will extend signatures to cover well-formedness of formulas by assigning signatures to logical connectives. In particular, RIF dialects would be entitled to introduce connectives, such as modal operators, which do not explicitly exist in RIF-FLD.

4 The Semantic Framework

The RIF-FLD semantic framework defines the notions of semantic structures and of models for RIF-FLD formulas. The semantics of a dialect is derived from these notions by specializing the following parameters.

1. The effect of the syntax.
   The syntax of a dialect may limit the kinds of terms that are allowed. For instance, if a dialect's syntax excludes frames or terms with named arguments then the parts of the semantic structures whose purpose is to interpret those types of terms become redundant.
2. Truth values.
   The semantic framework allows formulas to have truth values from an arbitrary partially ordered set of truth values, TV. A concrete dialect must select a concrete partially or totally ordered set of truth values. For instance, most dialects are expected to stay within the regular two-valued

category, but, for example, logic programming dialects that are based on the well-founded semantics would use a three-valued logic where the order is $true > undefined > false$.

3. Datatypes.
   A datatype is a set of symbols that have a fixed interpretation in any semantic structure. RIF-FLD defines a set of core datatypes that each dialect is required to include as part of its syntax and semantics. However, it does not limit dialects to just the core types: they can introduce additional datatypes, and each dialect must define the exact set of datatypes that it includes.
   This is just a remark in passing about the role of datatypes in RIF, which is beyond the scope of this survey. RIF datatypes are defined in a separate document produced by the working group [20].
4. Logical entailment.
   Logical entailment in RIF-FLD is defined with respect to an unspecified set of *intended models*. A RIF dialect must define which models are considered to be intended. For instance, one dialect might specify that *all* models are intended (which leads to classical first-order entailment), another may regard only the minimal models as intended, while a third might use only well-founded or stable models [22, 12].

We will not reproduce all the definitions here, but instead will highlight the most interesting aspects. The definition of semantic structures is pretty standard, especially to those who are familiar with F-logic and HiLog [17, 5]. The main differences are the mechanisms for dealing with multiple truth values (recall that the set TV of truth values can include more than the standard *true* and *false*) and formula reification. It amalgamates the techniques from [17, 5] to allow reification of frames.

Another interesting technique is used to define the semantics of document formulas. Recall that documents can import other documents, and documents can have local symbols. So, import is not just a mechanical union of all the imported document: the local symbols need to be disambiguated. FLD provides a model-theoretic semantics for that.

What makes FLD into a true framework for a range of different semantics is the concept of *entailment* that is based on the notion of intended models. To make the problem clear and highlight the difficulties, let us recall that apart from the syntax, what makes the different logic languages really different is their notion of entailment, i.e., the way they determine which formulas are regarded as consequences of other formulas. For instance, a large subset of first-order logic can be seen as a rule-based language. In such a language, the formula $p \leftarrow \neg p$ logically entails p, but not, say, q. If the same formula is considered to be part of a logic programming language with $\neg$ understood as default negation then the situation is different. First, there are several semantics for default negation, and two of them are widely used. According to the stable model semantics [12], $p \leftarrow \neg p$ is an inconsistent formula, so every conclusion follows, including q. According to the other popular semantics, the well-founded semantics [22], $p \leftarrow \neg p$ is consistent, but *nothing* of interest follows from it: neither p nor q.

The question therefore is: how does one accommodate all these different semantics in one framework so that the different RIF dialects could share the same machinery and be compatible with each other? The solution adopted in RIF-FLD was proposed by Shoham over two decades ago [21] when he observed that many logics that seemingly use completely different notions of entailment share essentially the same elements and can be explained away with the help of one simple definition.

We already talked about the notion of semantic structures, which is also often called *interpretation* in the literature. The purpose of semantic structures is to define certain sets and functions, which together determine the truth value (drawn from the set TV) of every well-formed formula in the logic language. If a semantic structure assigns the value *true* to a formula then it is said to be a *model* of that formula.

If $\mathbf{S}$ is a set of semantic structures then we say that one formula, ϕ, $\mathbf{S}$-entails another formula, ψ, if and only if for every semantic structure in $\mathbf{S}$, if it is an *intended* model of ϕ then it is also a model of ψ.

It turns out that all the interesting logic-based rule languages, including first-order logic and many others, define their notions of entailment in this or an equivalent way. The only difference is the set $\mathbf{S}$, which they consider in defining entailment, and what they consider to be an "intended" model. For instance, first-order logic has the simplest definition in this regard: $\mathbf{S}$ is just the set of all semantic structures and every model is intended. Other logics are more picky. For instance, $\mathbf{S}$ might contain only Herbrand semantic structures [19], and only minimal (in a certain sense) models might be considered as intended. Yet other languages have their own ideas about what is intended. We already mentioned the well-founded semantics and the stable-model semantics, for which the intended models are, as their names suggest, the well-founded models and the stable models, respectively [22, 12].

So, the bottom line is that RIF-FLD defines entailment with respect to the sets of intended models, as above, but it does not specify what these intended models are—it only defines semantic structures in general. It is left to the dialects to choose the appropriate notion.

5 The Basic Logic Dialect

The Basic Logic Dialect, RIF-BLD, is currently the only fully specified dialect of RIF. From the expressivity point of view, this dialect corresponds to the familiar Horn subset of logic programming [19]. No negation of any kind is allowed in the rule head and in the body. However, RIF-BLD has many syntactic extensions with respect to stock Horn rules. These include:

- Conjunctions in rule heads and disjunctions in rule bodies.
- Frames, membership, and subclass formulas.
- Predicates and functions with named arguments.
- Data types, group and document formulas.
- Equality both in rule heads and bodies.

There is also one notable restriction compared to FLD (and to many logic programming languages, like Prolog): as in a standard textbook version of first-order logic, every symbol is allowed to occur in at most one context in any document (including the imported documents). Thus, if a symbol occurs in the context of, say, binary predicate then it cannot occur as a ternary predicate. It cannot also occur as a function symbol or individual constant.

Ostensibly, these extensions and restrictions are supposed to simplify round-trippable translations to and from RIF-BLD (see Section 3.1 about round-tripping), but ultimately they are results of compromises. While they do simplify translation for some languages, they also make round-trippable translation harder for others. Nevertheless, round-tripping is helped greatly by another interesting feature of RIF: *meta-information*. In RIF-FLD (and in RIF dialects), meta-information can be attached to various syntactic objects at a very fine-grained level. For instance, it can be attached to variables, constants, etc. If enough meta-information is supplied with the RIF document obtained by translation from the language of some other system, then translation from that document back to the original system can be done unambiguously.

RIF-BLD can be easily defined as a specialization of the syntax and semantics of RIF-FLD. The restriction about the uniqueness of context for every symbol can be achieved by requiring that the signatures that are associated with the symbols that are used in RIF-BLD terms can have at most one arrow expression. Other syntactic restrictions are expressed by disallowing negation in rule implications and disjunction in rule heads. The corresponding semantic restrictions largely follow from the restrictions on the syntax. The exact details can be found in [1, Section 6].

6 Conclusions

This paper is an introduction to RIF Framework for Logic Dialects, an extensibility framework that ensures that the current and future dialects of the Rule Interchange Format share common syntactic, semantic, and XML markup apparatus. RIF-FLD is still work in progress: some details may change and additions to the framework should be expected.

Apart from RIF-BLD and the dialect under development for production rule systems, other dialects are being planned. These include the logic programming dialects that support well-founded and stable-model negation, a dialect that supports higher-order extensions as in HiLog [5], and a dialect that extends RIF-BLD with full F-logic [17] support (BLD accommodates only a very small part of F-logic).

The development of the RIF standard is an open process and feedback from experts and users is welcome. All the documents of the working group, meeting agendas, and working lists are publicly available at the group's Web site `http://www.w3.org/2005/rules/wiki/RIF_Working_Group`. The working version of the RIF framework document can be found at the following address: `http://www.w3.org/2005/rules/wiki/FLD`.

References

1. H. Boley and M. Kifer. RIF basic logic dialect. http://www.w3.org/TR/rif-bld/, October 2007.
2. H. Boley and M. Kifer. RIF Basic logic dialect. W3C Working Draft. `http://www.w3.org/TR/rif-fld/`, July 2008.
3. H. Boley and M. Kifer. RIF Framework for logic dialects. W3C Working Draft. `http://www.w3.org/TR/rif-fld/`, July 2008.
4. F. Bry, M. Eckert, and P.-L. Patranjan. Reactivity on the web: Paradigms and applications of the language xchange. *Journal of Web Engineering*, 5(1):3–24, 2006.
5. W. Chen, M. Kifer, and D.S. Warren. HiLog: A foundation for higher-order logic programming. *Journal of Logic Programming*, 15(3):187–230, February 1993.
6. K.L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 292–322. Plenum Press, 1978.
7. W.F. Clocksin and C.S. Mellish. *Programming in Prolog*. Springer-Verlag, 1981.
8. C. de Sainte Marie and A. Paschke. RIF Production rule dialect. W3C Working Draft. `http://www.w3.org/TR/rif-prd/`, July 2008.
9. M. Dean, D. Connolly, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P.F. Patel-Schneider, and L.A. Stein. Owl web ontology language 1.0 reference. Technical report, WWW Consortium, November 2002.
10. H.B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, 2001.
11. D.C. Fallside and P. Walmsley. XML Schema Part 0: Primer Second Edition. Technical report, WWW Consortium, October 2004. `http://www.w3.org/TR/xmlschema-0/`.
12. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Logic Programming: Proceedings of the Fifth Conference and Symposium*, pages 1070–1080, 1988.
13. M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3/4):365–386, 1991.
14. Drools. Web site. `http://labs.jboss.com/drools/`.
15. Jess, the rule language for the java platform. Web site. `http://herzberg.ca.sandia.gov/jess/`.
16. M. Kifer. FLORA-2: An object-oriented knowledge base language. The FLORA-2 Web Site. http://flora.sourceforge.net.
17. M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of ACM*, 42:741–843, July 1995.
18. A. Kozlenkov. *PROVA: A Language for Rule-based Java Scripting, Data and Computation Integration, and Agent Programming*, May 2005.
19. J.W. Lloyd. *Foundations of Logic Programming (Second Edition)*. Springer-Verlag, 1987.
20. A. Polleres, H. Boley, and M. Kifer. RIF Datatypes and built-ins. W3C Working Draft. `http://www.w3.org/TR/rif-dtb/`, July 2008.
21. Y. Shoham. Nonmonotonic logics: meaning and utility. In *Proc. 10th International Joint Conference on Artificial Intelligence*, pages 388–393. Morgan Kaufmann, 1987.
22. A. Van Gelder, K.A. Ross, and J.S. Schlipf. The well-founded semantics for general logic programs. *Journal of ACM*, 38(3):620–650, 1991.