Towards a Rule Interchange Language for the Web

François Bry[1] and Massimo Marchiori[2]

[1] University of Munich, Germany
http://www.pms.ifi.lmu.de/
[2] University of Venice, Italy, and W3C
http://www.w3.org/People/Massimo/

Abstract. This articles discusses rule languages that are needed for a a full deployment of the Semantic Web. First, it motivates the need for such languages. Then, it presents ten theses addressing (1) the rule and/or logic languages needed on the Web, (2) data and data processing, (3) semantics, and (4) engineering and rendering issues. Finally, it discusses two options that might be chosen in designing a *Rule Interchange Format* for the Web.

1 Introduction

This articles discusses rule languages that are needed for a a full deployment of the Semantic Web. First, this article motivates the need for such languages by considering four classes of (established or emerging) applications:

1. Business Rules,
2. Information Systems,
3. Negotiations, and
4. Ontologies.

Then, this article presents ten theses successively addressing:

1. the kinds of rule, or more generally logic, languages needed on the Web and Semantic Web,
2. data and data processing,
3. semantics, and
4. engineering and rendering issues.

Finally, this article discusses two options that might be chosen in designing a *Rule Interchange Format* for Web and Semantic Web applications.

Some of the views reported about in this article have been presented at the *W3C Workshop on Rule Languages for Interoperability* (27-28 April 2005, Washington, D.C., USA, http://www.w3.org/2004/12/rules-ws/) and in [6].

2 The Need for Rule Languages on the (Semantic) Web

Four classes of (already well established or currently emerging) applications can be mentioned for motivating the need for rule languages on the Web and on the Semantic Web. These application classes are:

- *Business Rules,* a first well-established and active class of applications.
- *Information Systems,* another well-established and active class of applications.
- *Negotiations,* especially for trust establishment, an emerging class of applications on the Web.
- *Ontologies,* another emerging class of applications on the Web.

In the following, these four classes of Web or Semantic Web applications are successively introduced and discussed.

2.1 Business Rules.

The Business Rule Group (BRG),[3] a well-known independent community of business and IT professional striving for standardizing concepts and formalisms used in the field, defines a "business rule" in the following very general terms:

> *"a statement that defines or constraints some aspects of business. It is intended to assert business structure or to control or influence the behaviour of the business."*

Business rules are often said to specify the "business logic" of a company and to exist, possibly implicitly, in every company. Business rule formalisms aim at making explicite a company's business logic. Thus, business rule formalisms serve not only the purpose of automation, but also of the purpose of a company's consciousness of its own working.

A more technical definition of "business rules" could be as follows. Business processes often refer to decision points at which conditions are evaluated and, depending on the evaluation, actions are performed. "Business rules" denote rule-based formalisms used to specify these points, conditions, and actions, as well as how they relate to each other.

Let us consider an example illustrating the concepts of "business logic" and "business rule". In a simplified form, *business rules* of a car rental company specifying 'price regulations for company customers', can be as follows: [4]

- No more than 20 cars can be rented at once by a (company) customer at a reduced company price.
- Reduced company prices are not applicable on Class A cars.

Beside regulations for company customers, the car rental company might have specified other business rules expressing 'general car renting regulations':

[3] `http://www.businessrulesgroup.org/brghome.htm`

[4] This example is inspired from the EU-Rent business rule use case [16].

- No more than one car can be rented at once by a person.
- Price reductions are not applicable to young drivers, i.e. drivers under 25 years.

Note that the above two sets of business rules, the 'price regulations for company customers' and the 'general car renting regulations' can well be at distinct Web sites, reflecting that two different departments of the car rental company at different locations are responsible for these two sets of business rules.

A certain company customer of the car rental company might also well have its own 'car rental regulations', e.g.:

- Junior employees are at most 24 years old.
- Every employee is junior or senior employee.
- Senior employees rent only Class A cars.

Note that the above regulations are expressible in OWL-DL but not in RDF, as quantifiers, cardinalities, and negation are needed.

Consider now the *forms of reasoning* that might be needed in processing business rules like the above regulations:

- To draw the conclusion that an employee older than 25 years is a senior employee, *excluded middle*[5] is needed.
- To draw the conclusion that no junior employees can rent a car on the (reduced) company price, *refutation*[6] and *monotonic negation*[7] are needed.
- To check whether a regulation, say *"No more than 20 cars can be rented at once by a (company) customer at a reduced company price."*, is currently enforced[8] *non-monotonic negation*[9] is needed.
- To determine so-called 'gold customers', i.e. customers renting cars at least 5 times during the last 12 months, *constructive reasoning* suffices, i.e. excluded middle and refutation are *not* needed.

Important aspects of business rules from a user's viewpoint are expressed in the BRG's *Business Rules Manifesto* [15]. In the following, those aspects relevant to this paper are briefly recalled:

- *Separation From Processes.* Business rules should be expressed separately from business processes.
- *Declarative, not Procedural.* Business rules should be expressed in declarative, not procedural formalisms.
- *For Business People.* Business rules should be expressed in formalisms easily understandable by "business people", i.e. easily understandable without knowledge of programming languages.

[5] At least one of A and $\neg A$ is true.
[6] If under the assumption A, a contradiction, i.e. B and $\neg B$ for some B, can be derived, then $\neg A$ is proven.
[7] The negation of classical logic and mathematics.
[8] For sure, such a regulation might will be violated from time to time!
[9] The negation used in concluding that flights not mentioned in a time table do not exist.

Depending on usages, three different kinds of business rules are usually considered, especially in the *Semantics of Business Vocabulary and Business Rules (SBVR)* [23]:

- *Structural Rules* that specify static "structural restrictions" or "constraints", e.g. *"a customers can rent at most one car at a time."* Structural rules amount to information systems' *integrity constraints.*
- *Derivation Rules* that specify how to derive additional data from available data, e.g. *a customer who has spent more than 1 000 $ in total premiums qualifies as "gold" customer."* Often, derivation rules involve arithmetical expressions, e.g. *"RentalCharge = Days × GroupRentalRate × (100 + DiscountPercentage) / 100 + PenaltyCharge"*. Derivation Rules amount to information systems' *deduction rules.*
- *Dynamic Rules* that express changes such as data updates, e.g. *"for a car rental from Switzerland, change the Euro price into a Swiss Franc price."*

Business rule applications often refer to "production rules". This rule concept is not further discussed in this section because it refers to the implementation layer. It is discussed below in Section 3.

The case study EU-Rent [16] developed by the Business Rule Group and [26] are good introductions to business rules and their use in applications.

2.2 Information Systems

Information systems, implemented using a Database management System or not, often refer to two kinds of rules:

- *Deduction Rules* that specify how to derive additional data from available data. Deduction rules amount to what is called *derivation rules* in the business rules community.
- *Integrity Constraints* specify constraints on data stored in an information system the violation which is worth being noted. Integrity constraints amount to what is called *structural rules* in the business rules community.

An information system could store student course attendances and examinations rankings. Deduction rules could be used to specify those students that are allowed to register to advance courses depending on courses they have formerly attended or of examinations they already have passed. Integrity constraints could express existential or disjunctive requirements like, e.g. *"every Computer science student should have a minor"* or *"every Computer science student should have as a minor Mathematics or Economics"*, or, e.g. negative requirements such as *"no Computer Science students can have Economics as a minor"*.

Deduction rules are often assumed to be "definite", i.e. to have consequents that are neither disjunctive nor existential. In such a case, neither excluded middle nor refutation is needed for drawing conclusions for deduction rules.

A widespread approach is to process integrity constraints like queries, i.e. to check whether integrity constraints are all enforced or if some integrity constraint

is violated. For efficiency reasons, it is a common approach to evaluate integrity constraints incrementally. Assuming that the information system enforces all integrity constraints in a first place, an incremental evaluation approach evaluated after each update of the information system only those (instances of) integrity constraints that might be violated by an update. If an integrity constraint states that *"no Computer Science students can have Economics as a minor"* and if a new Computer Science student, say Anna, is registered in the information system, then an incremental evaluation of the integrity constraint would consist in only checking that Anna does not have Economics as a minor.

2.3 Negotiations

Negotiations, especially for establishing trust, is an emerging issue of increasing importance on the Web. The following scenario, inspired from a use case of the W3C Rule Interchange format (RIF) Working Group [11], introduces into the issue.

A customer wants to buy a device at an eShop. The customer's and eShop's systems negotiate so as to automatically establish trust. The objective of the negotiation is to agree on payment modalities. The negotiation is based on both sides' policies, expressing via rules who both sides trust and for what purposes, and on credentials each system has.

When a customer wants to buy a device, eShop discloses to this customer's system a policy with alternatives like:

- A gold card holder is given a 10% discount on any purchase.
- An eShop employee gets 20% discount on devices of this type.
- Any other buyer must provide to the shop credit card information together with delivery information (address, etc.).
- The credit cards accepted are VISA and MasterCard.

The policy also states that for buyers providing a valid gold card or who are eShop employees, no further interactions and verifications are needed.

The policy also states in case credit card information is disclosed by a buyer, still the fulfillment of some other conditions might be required and/or still the purchase request might be denied. Thus, eShop policy includes rules such as:

- If the customer is in the eShop client blacklist, then deny purchase request.
- If the customer 's credit card is revoked. then deny purchase request.

Alice wants to buy a device at eShop.

Once Alice's negotiation system receives the eShop's policy, it checks Alice's credentials that are available and whether they fulfill eShop's policy.

Alice has a credit card but her own policy forbid to disclose it to everyone. Alice's system asks eShop to provide a proof of its membership to the Better Business Bureau (BBB), Alice's most trusted source of information on online shops.

eShop has such a credential and its policy is to release it to any potential purchaser. Therefore eShop releases it to to Alice's negotiation system. Alice's negotiation systems now checks that disclosing Alice's credit card to eShop would not violate Alice's denial constraints:

- Do not disclose two different credit cards to the same online shop.
- Never to disclose both Alice's birth date and postal code.

Alice's constraints are respected. Therefore, Alice's negotiation system discloses Alice's credit card information to eShop. eShop checks that Alice is not in its client black list, then confirms the purchase transaction, generates an email notification to the Alice giving information about the purchase, and notifies eShops's delivery department.

Rules used for negotiation have two interesting specificities:

- Several, typically two, parties are involved, each having its own set of rules.
- A party discloses its rules to other parties in a stepwise manner.

2.4 Ontologies

Ontologies, expressed e.g. in RDF or in OWL, give rise to specify the "logical structure" of application fields using more or less expressible logic languages. An ontology might be used to describe the services, e.g. cars and kinds of renting contracts, a car rental companies offers. An ontology might also be used to specify the regulations of a course of studies.

As a consequence, ontologies can be used for two different and complementary purposes.

First, an ontology can be used for investigating properties of the "logical structure" it specifies. For example, an ontology specifying the regulations of a course of studies could be used for analyzing the regulation: it consistency, i.e. freeness from contradictions like *"a Computer Science student must have minors in Mathematics and in Economics"* and *"no Computer Science student may have more than one minor"*, its implicit consequences, e.g. the minimal duration before a student is allowed to register for a certain examination, etc.

Second, an ontology can be used like a database schema or the type system of a program for describing the entities, or objects, an information system refers to. Let us refer to this use of an ontology as "specifying the logical structure of an information system."

Third, an ontology specifying the logical structure of an information system often also provides with integrity constraints that the information system is expected to enforce. This would, e.g., be the case of an ontology specifying the regulations of a course of studies stating that *"no Computer Science student may have more than one minor"*.

Fourth, an ontology specifying the logical structure of an information system often also provides with deduction rules for the information system. This would, e.g., be the case of an ontology specifying the regulations of a course of studies

stating that *"a Computer Science student must have minors in Mathematics and in Economics"*.

Reasoning with ontologies usually requires excluded middle and/or refutation.

3 Ten Theses on Logic Languages for the Semantic Web

3.1 Languages

Thesis 1 (Diversity) *The (Semantic) Web requires logic languages of different kinds:*

1. *three kinds of reasoning, or deductive, languages, viz.*
   (a) *constructive rules (or views),*
   (b) *normative rules (or integrity constraints),*
   (c) *descriptive specifications (or ontologies),*
2. *and reactive rules.*

Constructive rules,[10] called 'views' in databases, specify how to derive new data from data already available. Constructive rules typically involve data selection and grouping. Constructive rules are often, but not always, expressed as implications of the form `new-data` $\Leftarrow$ `query`. Examples of constructive rules are SQL views, Datalog or pure Prolog clauses,[11] and XSLT templates. Queries after XQuery can be seen as constructive rules with intertwined query and new-data parts. CSS rules can also be seen as constructive rules: CSS selectors are a kind of queries, declaration-blocks (or {}-blocks) specify how new, styled, data are constructed. RDFS semantic rules are further examples of constructive rules. Inference rules[12] used in specifying proof systems, are also constructive rules (*cf. infra* Thesis 8).

Normative rules, called 'integrity constraints' in databases, express conditions that data must fulfill, e.g. ISBN numbers uniquely characterize books, and that must be checked when data are updated. Data schemas, especially tree grammars in their various disguises, e.g. DTD, XML Schema, RelaxNG, etc., express normative rules.[13] Normative rules can be expressed as denials and evaluated like constructive rules. A denial is a rule of the form `false` $\Leftarrow$ `query` where the head `false`, or `error(...)`, etc., denotes a violation of a requirement `req` and the denial's body `query`expresses a negation of this requirement, i.e. `query` $\equiv \neg$`req`. E.g. the following denial expresses that ISBN numbers uniquely characterize book titles: `error(ISBN)` $\Leftarrow$ `book(Title1, ISBN)` $\wedge$ `book(Title2, ISBN)` $\wedge$ `Title1` $\neq$ `Title2`.

[10] The name stresses that consequences from such rules can be drawn in constructive logic, i.e. without relying on excluded middle or refutation.

[11] I.e. Prolog clauses without imperative predicates.

[12] E.g. modus ponens: If both A and $A \Rightarrow B$ are provable, then B is provable.

[13] However, variables in grammars differ from logic variables, since different occurrences of a same grammar variable represent different data instances.

Descriptive specifications specify data types and relationships between data types without necessarily referring to actual data. They are used in software specifications, data schemas, and ontologies. They are often expressed in logics[14] corresponding to classical logic fragments with *restricted quantifications* of the forms $\forall x : s\ F[x]$ and $\exists x : s\ F[x]$ restricting the variable x to some sort, class, entity, etc. s. Such quantifications can be expressed in classical logic as $\forall x\ s(x) \Rightarrow F[x]$ and $\exists x\ s(x) \wedge F[x]$, resp. using a conveniently defined unary predicate symbol s.

It is worth noting that, in many cases, the distinction between normative rules (integrity constraints) and descriptive specifications (ontologies) subtly depends on the use. Consider a system of rules expressing some regulation, e.g. under which conditions students are allowed to register for courses. In drawing conclusions from the regulation, or in verifying that it is consistent or non-redundant, the regulation is used as a descriptive specification – certain forms of reasoning such as excluded middle and refutation make sense and might even be indispensable. In verifying that student registrations to courses enforce the regulation, the regulation is used as integrity constraint – excluded middle and refutation do not make sense.[15]

Reactive rules specify how a data store can be modified depending on the current state of the store and, in some languages, on events. Reactive rules commonly have one of the forms `if condition then action` and `on event if condition then action`. Rules of the first kind are called *production rules*,[4] rules of the second, *ECA* (short for *Event-Condition-Action*) *rules*. In production and ECA rules, `condition` is an (atomic or compound) query to the data store similar to a body of a constructive or normative rule, and `action` is an atomic (i.e. single) or compound update of the data store (typically consisting of insertions, removal, and/or changes in a data item). In an ECA rule, `event` denotes an *event query*, i.e. a query to events received so far. An event query can be atomic, i.e. refer to a single event, or compound, i.e. refer to composite events. In the following, the condition of a production or ECA rule is called *standard query* so as to stress its similarity with the body of a constructive or normative rule.[16]

Thesis 2 (Negation) *Non-monotonic negation*[17] *is the negation of choice for constructive rules (views), normative rules (integrity constraints), and reactive rules. Monotonic negation may, but must not, be offered in constructive, nor-*

[14] E.g. sorted logics and description logics.

[15] One might object that Prolog, or a Prolog-like proof-system, can used for integrity checking, integrity constraints been expressed as denials, and that the proof method of Prolog, SLD resolution, is a refutation method. In fact, as opposed to general resolution, SLD resolution can be re-expressed in constructive logic [10], i.e., without referring to refutation.

[16] [18] further discusses how constructive and reactive rules, called 'passive' and 'active' resp., relate.

[17] The negation used in concluding that flights not mentioned in a time table do not exist.

mative, and reactive rules. Monotonic negation is the negation of choice for descriptive specifications (ontologies).

Non-monotonic negation, *cf.* [9] for selected articles, is the negation of choice for constructive rules (views) because data constructions depends on both, available and non-available data. Since normative rules can be expressed as constructive rules (*cf. supra* Thesis 1), non-monotonic negation is also the negation of choice for normative rules. Non-monotonic negation is the negation of choice for reactive rules, too, for both 'event queries' (i.e. the `event` parts of ECA rules) and 'standard queries' (i.e. the `condition` parts of production or ECA rules) refer to the presence or absence of data, events resp.

Monotonic negation is the negation of choice for descriptive specifications because descriptive specifications do not refer to actual data, e.g. the flights listed in a time table, but instead to meta-level specifications, e.g. conditions flights must fulfill, the negation needed in descriptive specifications does not have to refer to the absence or non-availability of such data.

Recall (*cf. supra* Thesis 1) that the same rule can be used as a normative specification (integrity constraint) or descriptive specification (ontologies). As a consequence, the choice of a negation semantics, monotonic or non-monotonic, does not necessarily depend on the syntax of negation.

Thesis 3 (Coherency and Inter-Operability) *Inter-operable logic languages of the various kinds should be striven for. Inter-operability is sustained by the following forms of coherency: syntax coherency, rendering coherency, reasoning coherency, and explanation coherency.*

Syntax coherency means that expressions from different languages with similar meanings are expressed similarly. *Rendering coherency* means that expressions from different languages are (visually or verbally) rendered (*cf. infra* Thesis 10) similarly, possibly using the same rendering methods or tools. *Reasoning coherency* means that similar forms of reasoning applied on different languages, e.g. for deriving new data using constructive rules, for computing the closure of RDF specifications, or for checking normative rules, are performed using similar reasoners. Reasoning coherency is desirable both for programmers and language design, and implementation. An important aspect of reasoning coherency is to have a common semantics for non-monotonic negation in constructive, normative, and reactive rule languages. *Explanation coherency* means that similar forms of reasoning are explained, by explanation tools, relaying on similar explanation paradigms. This is very important for user interaction (cf. Thesis 10) and for all advanced applications sitting in the top trust layer of the Semantic Web tower (think for example of large-scale expansions of PCA techniques *à la* [1]).

3.2 Data and Data Processing

Thesis 4 (Data Distribution and Versatility, and Meta-Level Reasoning) *A logic language for the Semantic Web must access data everywhere on the Web; be 'data versatile', i.e. capable of accessing data and meta-data in any*

common Web Semantic Web format – especially XML, RDF, Topic Maps, and OWL, as well as the formats of Semantic Web logic languages –, and capable of some forms of meta-level reasoning

There has already been a number of pleas in favour of data versatile query languages, e.g. [25]. Data versatility is consistent with the decentralized nature of the Web, and favours data aggregation, an important factor to consider in the overall picture of the Semantic Web.

Meta-level reasoning poses interesting, but not impossible, challenges. Meta-level reasoning has bad reputation among Computational Logicians, however, conveniently, e.g. constructively, restricted, *cf.* [8] meta-level reasoning is semantically as safe, and practically as useful as higher-order functions in Functional Programming. Note that meta-level reasoning is already present, though in a limited form, on the Semantic Web: RDF Schema, the "RDF Vocabulary Description Language", is itself an RDF Vocabulary for describing terms in an RDF vocabulary.

Thesis 5 (Reasoning Paradigms) *Constructive and normative rules (views and integrity constraints) should be evaluable by both forward chaining*[18] *and backward chaining*[19]*, backward chaining being the reasoning paradigm of choice. Descriptive specifications (ontologies) call for (non-constructive) reasoning, including excluded middle*[20]*, non-contradiction*[21] *and refutation*[22]*. The reasoning paradigms of Semantic Web logic languages should support grouping, aggregation, theory reasoning, and non-monotonic negation.*[23]

On the Web, forward chaining is well-suited only for well-defined and closed sets of Web sites. Queries referring directly, or indirectly (through sub-queries triggered by constructive rules at queried Web sites) to a set of Web sites that cannot be statically[24] recognized, cannot be evaluated by forward chaining. Indeed, with such queries, forward chaining would require to compute intermediate results from all possible Web sites.Thus, on the Web, backward chaining is the reasoning paradigm of choice for constructive and normative rules.

Theory reasoning, a term coined after Mark Stickel's 'theory resolution' [27], denotes enhancing a general purpose reasoning method with special reasoners where convenient, e.g., reasoning on bank accounts with a basic arithmetic 'theory reasoner' instead of the Peano axioms of Arithmetic.

Thesis 6 (Event Processing) *Event broadcasting is undesirable on the Web. Events can be exchanged between Web sites using a push, or a pull model. Pushed events can be sent as data streams, calling for streamed query evaluation methods.*

---

[18] Also called bottom-up reasoning.
[19] Also called top-down reasoning.
[20] At least one of A and $\neg A$ is true.
[21] At most one of A and $\neg A$ is true.
[22] If under the assumption A, a contradiction, i.e. B and $\neg B$ for some B, can be derived, then $\neg A$ is proven.
[23] Preferably with a semantics understandable without PhD in Logic!
[24] I.e. before query evaluation.

Evaluating event queries, e.g. the event parts of ECA rules, calls for event driven query evaluation methods.

On the Web, events can not be broadcasted, i.e. indiscriminately sent to all sites, because this would result in too high a traffic. Events can be exchanged on the Web sites via either push, i.e. events are sent by the emitters to specific recipients, or pull methods, i.e. each site publishes the events it emits, together with the event's recipients, on a 'blackboard' which is repeatedly queried by the potential recipient sites. Such queries are called *continuous*. With the push model, event can be sent as 'data streams' [5]. Continuous queries [29, 2, 22, 24], data streams [5], and event queries [7, 3] require specific query evaluation methods.

3.3 Semantics

Thesis 7 (Declarative Semantics) *Logic languages for the Semantic Web, except reactive rule languages, should have declarative semantics defined as 'Tarski-style model theories'.*

Tarski-style models [17], i.e., the models of classical logic, are expressed in terms of so-called 'valuation functions' that are defined recursively on a formula's structure. They make possible to evaluate a formula independently of other formulas. Therefore, they are easy to understand, and they do not require complex operational semantics.[25]

Production and ECA rules amount to *imperative programming*, hence they are inherently not amenable to declarative semantics. However, (1) declarative semantics are possible and desirable for the 'standard query' and 'event query' languages used in production or ECA rules languages, and (2) a formal semantics amenable to reasoning on production and ECA rule programs is possible (and desirable!).

Thesis 8 (Operational Semantics) *The operational semantics of a logic language is conveniently expressed with constructive and normative rules. Backtracking is useful for a fine tuning of proof construction in implementing logic languages.*[26]

The operational semantics of a logic language or reasoner is usually and conveniently expressed in terms of inference rules of the form:

$$\frac{\texttt{Premise}_1 \ldots \texttt{Premise}_n}{\texttt{Conclusion}}$$

[25] Note that most declarative semantics for non-monotonic negation that do not assume stratified, or stratifiable, rules, e.g. the stable [14] and well-founded [13] semantics, do *not* have Tarski-style model theories.

[26] Backtracking is however undesirable as a programming concept for high-level logic languages like the logic languages needed on the Semantic Web because it destroys the language's declarativity. The operational paradigm(s) desirable for a Semantic Web logic languages can be equivalently called 'backtracking-free logic programming' or 'set-oriented functional programming'. It is worth noting almost of the query languages proposed for RDF are of this kind.

Inference rules can be seen as constructive rules in a meta-language specifying proofs for formulas of the object-level language. Thus, constructive rules are subjacent to (the procedural semantics of) *every* rule language and reasoners. This observation has led to successful uses of the run-time system [28] of Prolog or of the Prolog language itself [19] for implementing efficient theorem provers. Normative rules, too, are convenient in specifying the procedural semantics of rule languages and reasoners for expressing constraints on the proof, or search, space. Reactive rules can be convenient in implementing logic languages and reasoners.[27]

3.4 Engineering and Rendering

Thesis 9 (Language Engineering) *Logic languages for the Semantic Web should be referentially transparent, strongly closed, have Web formats, and modern type systems.*[28] *The specification of abstract machines should be striven for.*

Referential transparency, i.e. within a same declaration scope two occurrences of a same expression have the same meaning, is desirable because it is *the* trait of declarativity. *Closure*, i.e. the data returned by a program are like, e.g. have formats similar to, the data accessed by programs in the same language. *Strong closure* means that the data returned by a program can be further processed by this same program. Strong closure is desirable because it eases structuring programs in sub-programs. *Web formats*, especially XML formats such as RuleML formats, are desirable for rule languages because they eases inter-changing programs on the Web, e.g., for Web services applications. *Abstract data types* and *static type checking* are desirable for Semantic Web reasoning and reactive languages as they are for any other programming languages: *"Well typed programs do not go wrong."* [21] *Abstract machines* are desirable because they are essential for wide-spreading languages.

Thesis 10 (Visual and Verbal Rendering) *Logic languages for the Semantic Web should have visual and verbal renderings.*

Declarative languages are especially well-suited to visual rendering and visual rendering is very appealing to potential users of logic languages for the Semantic Web, as the many systems for graphical rendering and/or visualization of business rules amply demonstrate.

Programs used on the Web and Semantic Web should be *verbalizable*, i.e. the rules or formulas they consist of should be expressible in a controlled language [20, 30, 12], i.e. in a non-ambiguous language resembling natural language. Rules,

[27] Since constructive and reactive rule languages can be used in specifying and implementing logic languages and reasoners, some claim that a single language of such a kind would be sufficient for the Semantic Web. This amounts to claiming that only one single, e.g., imperative, programming language could be sufficient for developing software.

[28] I.e., type systems supporting abstract data types and offering static type checking, parametric polymorphism, and modules.

e.g. expressing policy specifications and trust, verbalized in a controlled language would considerably help wide-spreading the (verbal as well as non-verbal forms of the) languages they are expressed in. And in fact, the importance of this concept has been further substantiated along these years by the fact that most of the commercially available business rule systems (who pay special attention to key factors like user base and commercial spreading) did evolve so to provide at least some basic form of verbalization.

4 Towards a Rule Interchange Language: 'Esperanto' or 'Lingua Franca'?

The Web makes it possible to distribute applications like, e.g, those mentioned above in Section 2. This calls for a formalism for transferring rules, e.g. business rules, integrity constraints, or policies, between Web sites. In designing such a "rule interchange language", we can identify some crucial questions are as follows:

- Should a rule interchange language be limited to a syntax?
- Should a rule interchange language have a formal declarative semantics?
- Should a rule interchange language have an operational semantics?

In this section, these questions are addressed and a, possibly controversial, position is taken by the authors.

Clearly, a rule interchange language must have a formally specified and unambiguous syntax for, otherwise, it could not be used for safely transferring rules between systems.

Clearly, a rule interchange language must also have a meaning, i.e. a declarative semantics, known to and understood by its users. Otherwise, it could hardly be used for transferring informations like the business rules, deduction rules, or integrity constraints of Section 2, between distributed users and systems. Here, interoperability is the key: without a formal semantics, "closed box" solutions are of course possible, based for example on the Web Services infrastructure to transport and trigger the rule language. But doing this, we lose interoperability, which is the key to success that justifies the same concept of a rule interchange language. Regarding the meaning of such a rule interchange language, it could be either an informal semantics, as formalisms like, e.g. UML have, or a formal semantics. One might think that an informal semantics is easier to understand and use because it is not based on formal, e.g. mathematical or logical formalisms. In fact, as programming languages amply demonstrate, a formal syntax considerably eases the use of a language.

Whether a rule interchange language should have an operational semantics is a question, which is currently much debated. A rule interchange language with a formal semantics could be used on the Web everywhere rules are to be processed. This could be done either by a direct processing, i.e. by using the rule language everywhere on the Web, or by translating from this language into other rule

languages locally used. The rule interchange language would be full-fledge rule language, a kind of 'Esperanto' of rule processing on the Web.

A serious drawback of an Esperanto of rule-based reasoning on the Web is its acceptance. Like Esperanto, the rule language might be well conceived, its use might remain limited to a small community of aficionados. Admittedly, a full-fledge language aiming at replacing existing rule languages might be rejected by the user communities of other rule languages.

Instead of an 'Esperanto' of rule processing on the Web, a 'Lingua Franca' might be preferable. The analogy is used here for denoting a rule language with a clear semantics, preferable specified in a formal manner, but without operational semantics. Of course, such a 'Lingua Franca' of rule interchange would require some additional treatment, i.e. to be converted into a processable rule language (possibly supporting also sublanguage profiling, a feature successfully employed for instance in OWL and XQuery). But it would make much sense, as the examples of rules mentioned above in Section 2 amply demonstrate: These examples are perfectly understandable, because one can give them a meaning. A formally specified declarative semantics is all what is needed for making possible a semantically safe interchange of rules, at the highest possible level, therefore better surviving the evolutionary tide of all the forthcoming concrete rule languages with their own operational semantics.

Acknowledgments.

The authors thank their colleagues from REWERSE (*cf.* `http://rewerse.net`), from the W3C Rule Working Group (*cf.* `http://www.w3.org/2005/rules/`), Michael Eckert, Tim Furche, Paula-Lavinia Pătrânjan, and Inna Romanenko (all four from the University of Munich) for interesting discussions on the topic of this article and useful hints and suggestions.

This research has been funded by the European Commission and by the Swiss Federal Office for Education and Science within the 6th Framework Programme project REWERSE number 506779 (*cf.* `http://rewerse.net`).

References

1. Andrew W. Appel and Edward W. Felten. Proof-Carrying Authentication. In *Proc. 6th ACM Conference on Computer and Communications Security*, 1999.
2. Shivnath Babu and Jennifer Widom. Continuous Queries over Data Streams. *SIGMOD Record*, 2001.
3. James Bailey, François Bry, and Paula-Lavinia Pătrânjan. Composite Event Queries for Reactivity on the Web. In *Proc. 14th Int. World Wide Web Conference*, 2005.
4. Lee Brownston, Robert Farrell, Elaine Kant, and Nancy Martin. *Programming Expert Systems in OPS5: An Introduction to Rule-based Programming.* Addison-Wesley, 1985.

5. François Bry, Fatih Coskun, Serap Durmaz, Tim Furche, Dan Olteanu, and Markus Spannagel. The XML Stream Query Processor SPEX. In *Proc. 21st Int. Conf. on Data Engineering (ICDE)*, 2005.
6. François Bry and Massimo Marchiori. Ten Theses on Logic Languages for the Semantic Web. In *Proc. of the Third Int. Workshop on Principles and Practice of Semantic Web Reasoning (PPSWR)*, LNCS 3703. Springer-Verlag, 2005.
7. François Bry and Paula-Lavinia Pătrânjan. Reactivity on the Web: Paradigms and Applications of the Language XChange. In *Proc. 20th Annual ACM Symp. Applied Computing (SAC)*, 2005.
8. Weidong Chen, Michael Kifer, and David Scott Warren. HILOG: A Foundation for Higher-Order Logic Programming. *Jour. of Logic Programming*, 15(3):187–230, 1993.
9. Jürgen Dix, Luís Moniz Pereira, and Teodor C. Przymusinski., editors. *Selected Papers from the Non-Monotonic Extensions of Logic Programming*. LNCS 1216. Springer-Verlag, 1996.
10. K. Doets. *From Logic to Logic Programming*. MIT Press, 1994.
11. W3C Rule Interchange format (RIF) Working Group. Automated Trust Establishment for eCommerce, November 2005. http://www.w3.org/2005/rules/wg/wiki/Automated_Trust_Establishment_for_eCommerce.
12. Norbert E. Fuchs, Uta Schwertel, and Rolf Schwitter. Attempto Controlled English – Not Just Another Logic Specification Language. In *Proc. 8th Int. Workshop (LOPSTR)*, LNCS 1559. Springer-Verlag, 1999.
13. Allen Van Gelder, Kenneth A. Ross, and John S. Schlipf. The Well-Founded Semantics for General Logic Programs. *Jour. ACM*, 38(3):620–650, 1991.
14. Michael Gelfond and Vladimir Lifschitz. The Stable Model Semantics for Logic Programming. In *Proc. Int. Conf. and Symp. Logic Programming*, 1988.
15. Business Rule Group. The Business Rules Manifesto, November 20003. http://www.businessrulesgroup.org/first_paper/br01c0.htm.
16. Business Rule Group. Eu-rent case study, 2001, 2005. http://www.eurobizrules.org/ebrc2005/eurentcs/eurent.htm and http://www.businessrulesgroup.org/first_paper/br01ad.htm.
17. Jerome Keisler. *Handbook of Mathematical Logic*, chapter Fundamentals of Model Theory, pages 47–103. North-Holland, 1989.
18. Rainer Manthey. Active and Passive Rules in Database Systems: How do They Relate. In *Proc. 1st Workshop on Advances in Databases and Information Systems*, 1994.
19. Rainer Manthey and François Bry. SATCHMO: A Theorem Prover Implemented in Prolog,. In *Proc. 9th Conf. on Automated Deduction*, 1988.
20. Massimo Marchiori and Janne Saarela. Query + Metadata + Logic = Metalog. In *Proc. QL '98, The Query Languages Workshop*, 1998. http://www.w3.org/TandS/QL/QL98/.
21. Robin Milner. Fuly Abstract Models of Typed λ-Calculi. *Theoretical Computer Science*, 4(1):1–22, 1977.
22. Benjamin Nguyen, Serge Abiteboul, Gregory Cobena, and Mihai Preda. Monitoring XML Data on the Web. In *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, 2001.
23. Object Management Group (OMG). Semantics of Business Vocabulary and Business Rules (SBVR), August. Revised Submission, http:/www.omg.org/docs/bei/05-08-01.pdf.

24. Sandeep Pandey and and Soumen Chakrabarti Krithi Ramamritham. Monitoring the Dynamic Web to Respond to Continuous Queries. In *Proc. 12th Int. World Wide Web Conference*, 2003.
25. Jonathan Robie. The Syntactic Web: Syntax and Semantic on the Web. In *Proc. XML Conf. and Exposition*, 2001.
26. Inna Romanenko. Use Cases for Reactivity on the Web: Using ECA Rules for Business Process Modeling. Master's thesis, Institute for Informatics, University of Munich, Germany, 2006. http://www.pms.ifi.lmu.de/publikationen/index.html#DA_Inna.Romanenko.
27. Mark E. Stickel. Automated Deduction by Theory Resolution. *Jour. of Automated Reasoning*, 1(4):333–355, 1985.
28. Mark E. Stickel. A Prolog Technology Theorem Prover: Implementation by an Extended Prolog Computer. *Jour. of Automated Reasoning*, 1988.
29. Douglas Terry, David Goldberg, David Nichols, and Brian Oki. Continuous Queries over Append-Only Databases. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 1992.
30. W3C. The Metalog Project. http://www.w3.org/RDF/Metalog/.