

## **Representing E-Business Rules for the Semantic Web: Situated Courteous Logic Programs in RuleML**

Benjamin N. Groszof

MIT Sloan School of Management

bgroszof@mit.edu ; <http://www.mit.edu/~bgroszof/>

Room E53-317, 50 Memorial Drive, Cambridge, MA 02142, USA

### **Abstract**

We give an overview of current efforts to standardize e-business rules knowledge representation (KR) in XML as part of the Semantic Web. We focus especially on the design approach and criteria of RuleML, an emerging standard that we are helping to lead. We discuss the issues of standardization and Webizing which RuleML addresses. We extend, for the first time, RuleML's definition from the ordinary logic programs KR to situated courteous logic programs (SCLP), an expressively general KR that supports prioritized conflict handling and procedural attachments for actions and queries. We give an overview of our prototype **SWEET (Semantic WEB Enabling Technology)**, a set of tools which enable, for the first time, communication and inferencing of e-business rules represented in SCLP RuleML. We illustrate SCLP RuleML by giving example rulesets from the realm of e-commerce business policies.

---

\*This paper is extended and updated from the version appearing in Proceedings of the Workshop on Information Technologies and Systems (WITS '01) (<http://www.busi.mun.ca/parsons/wits2001/>), held Dec. 15–16, 2001, New Orleans, LA, USA, in conjunction with the International Conference on Information Systems (ICIS '01). The system described here was a (refereed) software system demonstration at WITS '01, as well.)

# 1 Introduction

In this paper, we give an overview of our current efforts to standardize rules knowledge representation (KR) in XML. We focus especially on the design approach and criteria of RuleML (short for “Rule Markup Language”), an emerging standard that we are helping to lead. The Rule Markup Language Initiative<sup>1</sup> is part of the R&D community’s effort to develop the *Semantic Web*<sup>2</sup>, a vision of moving the Web to support program-to-program communication of data that has high-level semantics that are shared by both parties (programs) in communication. RuleML is based on a fundamental rule KR, declarative *logic programs* (LP). In previous work [7] [2] [4] [3] [5] [6] we have expressively extended ordinary LP (OLP)<sup>3</sup> with features for prioritized conflict handling and procedural attachments to perform actions and queries; the result is called *situated courteous logic programs* (SCLP). In previous work we have also developed BRML, an XML syntax for SCLP. We discuss BRML’s limitations – it only shallowly *Webized* SCLP.

The novel contributions of this paper are to extend RuleML’s specification to the SCLP case, to overview SCLP RuleML and the issues of standardization and *Webizing* which it addresses, and to overview the first prototype toolset for SCLP RuleML – which we have built.

Our prototype toolset is called **SWEET**, short for “**Semantic Web Enabling Technology**”. SWEET enables communication and inferencing of e-business rules represented in RuleML. We have a running first version of SWEET which was demonstrated at the Workshop on Information Technology and Systems (WITS '01) on Dec. 16, 2001.

---

<sup>1</sup>see <http://www.dfki.de/ruleml> and the author’s home page (<http://www.mit.edu/bgrosf>)

<sup>2</sup>see the overview of the Semantic Web Activity on the World Wide Web Consortium’s website <http://www.w3.org>

<sup>3</sup>sometimes called “general” LP or “normal” LP; see [1] for a helpful review

## 2 RuleML and expressively extended Logic Programs

We are leading, with Harold Boley of DFKI (Germany) and Said Tabet of Nisus Inc. (USA), an early-phase standards effort on a markup language for exchange of rules in XML, called RuleML (Rule Markup Language)<sup>4</sup>. The goal of this effort is eventual adoption as a Web standard, e.g., via the World Wide Web Consortium (W3C) within its new Semantic Web Activity.<sup>5</sup> Along the way there are a number of interesting new research issues.

RuleML is, at its heart, an XML syntax for rule knowledge representation (KR), that is inter-operable among major commercial rule systems. It is especially oriented towards four currently commercially important (“CCI”) families of rule systems: SQL (relational database), Prolog, production rules (cf. OPS5, CLIPS, Jess) and Event-Condition-Action rules (ECA). These kinds of rules today are often found embedded in systems built using Object-Oriented (OO) programming languages (e.g., C++ and Java), and are often used for business process connectors / workflow. These four families of rule systems all have common core abstraction: declarative logic programs (LP) ([1] provides a helpful review). “Declarative” here means in the sense of KR theory.<sup>6</sup> Note that this supports both backward inferencing and forward inferencing.

RuleML is actually a family (lattice) of rule KR expressive classes<sup>7</sup>: each with a DTD (syntax) and an associated KR semantics (KRsem). These expressive classes form a generalization hierarchy (lattice). In addition to specifying XML syntax for rules, RuleML specifies an associated KR semantics (KRsem).

---

<sup>4</sup><http://www.dfki.de/ruleml> and <http://www.mit.edu/~bgrossof/#RuleML>

<sup>5</sup><http://www.w3.org/2001/sw>

<sup>6</sup>in which: a given set of premises entails a set of sanctioned conclusions, independent of inferencing control strategy or procedural aspects, e.g., independent of whether inferencing direction is goal-directed query answering (“backward”) vs. data-driven (“forward”).

<sup>7</sup>“Class” here means an expressive subset of a logical language.

The KRsem specifies what set of conclusions are sanctioned for any given set of premises. Being able to define an XML syntax is relatively straightforward. Crucial is the semantics (KRsem) and the choice of expressive features.

The motivation to have syntax for several different expressive classes, rather than for one most general expressive class, is that: precision facilitates and maximizes effective interoperability, given heterogeneity of the rule systems/applications that are exchanging rules.

The kernel representation in RuleML is: Horn declarative logic programs. Extensions to this representation are defined for several additional expressive features:

- negation: negation-as-failure and classical negation;
- prioritized conflict handling: e.g., cf. *courteous* logic programs [7];
- disciplined procedural attachments for queries and actions: e.g., cf. *situated* logic programs [2];

and other features as well. In addition, RuleML defines some useful expressive restrictions (e.g., Datalog, facts-only, binary-relations-only), not only expressive generalizations.

In January 2001, we released a first public version of a family of DTD's for several flavors of rules in RuleML. This was presented at the W3C's Technical Plenary Meeting<sup>8</sup> held Feb. 26 to Mar. 2, 2001. Especially since then, RuleML has attracted a considerable degree of interest in the R&D community. Meanwhile, the design has been evolving to further versions.

RuleML largely grows out of the design approach and design criteria of Business Rules Markup Language (BRML) which was developed in our previous work at IBM Research and which is implemented in IBM CommonRules<sup>9</sup>

---

<sup>8</sup>a large convocation of most of its face-to-face standards working group meetings

<sup>9</sup><http://www.research.ibm.com/rules> and <http://alphaworks.ibm.com>

available under free trial license from IBM alphaWorks. The design approach and design criteria of CommonRules and BRML are described in [7], and in the documentation in the CommonRules download package. BRML's expressive class is situated courteous logic programs, i.e., declarative logic programs with negation-as-failure, (limited) classical negation, prioritized conflict handling, and disciplined procedural attachments for queries and actions.

SCLP and BRML were developed in large part to surmount the limitations of the first major attempt at a KR interlingua: Knowledge Interchange Format (KIF)<sup>10</sup>. KIF was developed in the early 1990's as a system for researchers, as opposed to commercial applications, to exchange logical-form knowledge. KIF's KR is essentially classical logic. It has not yet become widely used for deployed commercial applications. In particular, it has two major limitations that prevent it from representing e-business rules of the kind used in CCI rule applications. Firstly, it is *pure-belief*; it cannot represent/specify procedural attachments for queries or actions (or for anything else!). Secondly, it is *logically monotonic*; it cannot represent/specify negation-as-failure or prioritized conflict handling which are logically *non-monotonic*.<sup>11</sup> Yet procedural attachments and logical non-monotonicity are heavily used in CCI rule systems and their applications. Example kinds of non-monotonicity include: priority between rules in Prolog based on static rule sequence; dynamically-computed priorities among rules in production rule and ECA rule systems; inheritance with exceptions; and updating in databases (where more recent assertions override previous ones).

Another advantage of (situated courteous) logic programs is computational scaleability: inferencing is tractable (worst-case polynomial-time) for a broad expressive case: e.g., when the number of logical variables per rule is bounded,

---

<sup>10</sup><http://logic.stanford.edu/kif> and <http://www.cs.umbc.edu/kif/>

<sup>11</sup>A particular KR is said to be logically monotonic when it has the property that adding premises never results in retracting (sanctioned) conclusions.

and logical functions are disallowed; classical logic. By contrast, classical logic inferencing is NP-hard for this case.

RuleML differs in several significant respects from its BRML predecessor, however. it defines a family of DTD's. More deeply, however, These differences largely revolve around "Webizing" the KR:

- URI's<sup>12</sup> for logical vocabulary and knowledge subsets
- labels for rules/rulebases, import/export
- headers: meta-data describes the XML document's expressive class
- procedural attachments using Web protocols/services; queries or actions via CGI/servlets/SOAP/...

Such Webizing, and interoperability of KR on the Web, involve several kinds of practical mechanics beyond the representation proper. These include to:

- build on existing W3C standards: namespaces, ...
- share mechanisms with other emerging/extant standards for KR and ontologies on the Web, including especially RDF/RDFS<sup>13</sup> and DAML+OIL<sup>14</sup>, the ontology representation that is the main technical point of departure for the newly-formed WebOnt Working Group of the W3C
- use ontologies for rules, and rules for ontologies
- support ontology tags in: rulebase, predicate symbol, ...

---

<sup>12</sup>Uniform Resource Identifiers, a generalization of Uniform Resource Locators (URL's), a W3C standard

<sup>13</sup><http://www.w3.org>, search for Resource Description Format (RDF) and RDF Schema

<sup>14</sup><http://www.daml.org>

RuleML has some first steps of Webizing rule KR, including:

- URI's for logical vocabulary and knowledge subsets: predicates, functions, rules, rulebases, labels for rules and rulebases
- facilitation of an (alternative) RDF syntax: by having its XML syntax mostly avoid reliance on ordered-ness of child elements within any element (there are already some partial first drafts of such an alternative RDF syntax)
- support of object-oriented style argument "roles" (cf. named members of a Java/C++ class; rather than simply relying on position within an ordered tuple of arguments in the manner of, say, Prolog)

### 3 Example: Ordering Lead Time

#### Example 1 (Lead Time)

In business-to-business commerce, e.g., in manufacturing supply chains, sellers often specify how much lead time, i.e., minimum advance notice before scheduled delivery date, is required in order to place or modify a purchase order. An example of a parts supplier vendor's lead time policy is:

- (Rule A:) "14 days lead time if the buyer is a preferred customer."
- (Rule B:) "30 days lead time if the ordered item is a minor part."
- (Rule C:) "2 days lead time if: the ordered item's item-type is backlogged at the vendor, and the order is a modification to reduce the quantity of the item, and the buyer is a preferred customer."
- (Priority Rule:) "If Rules A and C both apply, then Rule C 'wins', i.e., 2 days lead time."

The rationale for Rule C is that the vendor is having trouble filling its overall orders (from all its buyers) for the item, and thus is pleased to have

this buyer relieve the pressure.

Rules A, B, and C may conflict: two or three of them might apply to a given purchase order. The Priority Rule provides partial prioritization information – its rationale might be that Rule C is more specific, or more recent, than Rule A. However, the above rule-set leaves unspecified how to resolve conflict between Rules A and B, for example; no relative priority between them is specified as yet. This reflects a common situation when rules accumulate over time, or are specified by multiple authors: at any given moment during the process of incremental specification, there may be insufficient justified priority to resolve all potential conflicts.

### Example 2 (Ordering Lead Time, in CLP)

The above example can be straightforwardly represented in CLP as follows<sup>15</sup>:

- ⟨a⟩ *orderModificationNotice*(?Order, days14)  
    ← *preferredCustomerOf*(?Buyer, ?Seller) ∧  
       *purchaseOrder*(?Order, ?Buyer, ?Seller).
- ⟨b⟩ *orderModificationNotice*(?Order, days30)  
    ← *minorPart*(?Order) ∧  
       *purchaseOrder*(?Order, ?Buyer, ?Seller).
- ⟨c⟩ *orderModificationNotice*(?Order, days2)  
    ← *preferredCustomerOf*(?Buyer, ?Seller) ∧  
       *orderModificationType*(?Order, reduce) ∧  
       *orderItemIsInBacklog*(?Order) ∧  
       *purchaseOrder*(?Order, ?Buyer, ?Seller).

The rule labels, e.g., *a*, at the left of each rule above are used as handles/names for specifying *prioritization* (partial-) ordering via the syntactically reserved predicate *overrides* which indicates that its first argument is

---

<sup>15</sup>This CLP example is taken from [7]; its longer continuations in the rest of this paper are novel.

higher priority than its second argument. *overrides* is otherwise an ordinary predicate, e.g., *overrides* facts can be inferred via rules.

*overrides(c, a).*

The scope of what constitutes conflict is specified by *mutual exclusion (mutex) statements*, which can be viewed as a kind of integrity constraint. Each such statement says that is a contradiction/inconsistency for a particular pair of literals to be inferred, given another particular condition. E.g., the following specifies that it is a contradiction to conclude two different values of the ordering-lead-time for the same order. The CLP KR's semantics enforce that the set of sanctioned conclusions respects (i.e., is consistent with) all the mutex's within the given CLP.

$$\begin{aligned} \perp \leftarrow & \text{orderModificationNotice}(\text{?Order}, \text{?X}) \wedge \\ & \text{orderModificationNotice}(\text{?Order}, \text{?Y}) \\ & | (\text{?X} \neq \text{?Y}). \end{aligned}$$

To represent this example directly as an Ordinary LP — while handling conflict appropriately in regard to priorities and guaranteeing consistency — requires modifying the rules to add extra “interaction” conditions that prevent more than one rule applying to a given purchase order situation. Moreover, adding a new rule requires modifying the other rules to add additional such interaction conditions. This is typical of conflicting rule-sets and underscores the advantage of the prioritized conflict handling expressive feature for modularity and ease of modification.

Next, we give a direct Ordinary LP version of Example 1 (i.e., without the courteous expressive features) that essentially behaves equivalently semantically to the Courteous LP version (Example 2). I.e., it entails the same conclusions about *orderModificationNotice*, for various cases/scenarios.

The following, for variety, is in an ASCII plain-text syntax for OLP<sup>16</sup>.

---

<sup>16</sup>which is used in IBM CommonRules and called there “the CLPfile syntax”

“FNEG” stands for  $\sim$ , i.e., for negation-as-failure.

Note that this representation is clumsier and lower-level than the courteous one, especially in regard to updating and avoiding overlap cases where the rules might conflict.

### Example 3 (Direct OLP Version of Lead-Time Example)

Vendor’s rules that prescribe how buyer must place or modify an order:

- A1) 14 days ahead if the buyer is a qualified customer, and  
the item is NOT a minor part, and  
the item is NOT backlogged.
- A2) 14 days ahead if the buyer is a qualified customer, and  
the item is NOT a minor part, and  
the modification is NOT to reduce the quantity of the item.
- B) 30 days ahead if the ordered item is a minor part, and  
the buyer is NOT a qualified customer
- C) 2 days ahead if the ordered item’s item-type is backlogged at the vendor,  
the order is a modification to reduce the quantity of the item, and  
the buyer is a qualified customer, and  
the ordered item is NOT a minor part.

/\*

-----

\*/

/\* Ordinary logic program rules follow.

Compared to the courteous logic program version, this

- omits priorities (overrides facts) and rule labels,

- uses FNEG’d interaction conditions

to avoid overlap cases where the rules might conflict.

\*/

```

orderModificationNotice(?Order,days14)
    <- preferredCustomerOf(?Buyer,?Seller) AND
        purchaseOrder(?Order,?Buyer,?Seller) AND
        FNEG minorPart(?Order) AND
        FNEG orderItemIsInBacklog(?Order) .

orderModificationNotice(?Order,days14)
    <- preferredCustomerOf(?Buyer,?Seller) AND
        purchaseOrder(?Order,?Buyer,?Seller) AND
        FNEG minorPart(?Order) AND
        FNEG orderModificationType(?Order,reduce) .

orderModificationNotice(?Order,days30)
    <- minorPart(?Order) AND
        purchaseOrder(?Order,?Buyer,?Seller) AND
        FNEG preferredCustomerOf(?Buyer,?Seller) .

orderModificationNotice(?Order,days2)
    <- preferredCustomerOf(?Buyer,?Seller) AND
        orderModificationType(?Order,reduce) AND
        orderItemIsInBacklog(?Order) AND
        purchaseOrder(?Order,?Buyer,?Seller) AND
        FNEG minorPart(?Order) .

```

Next, we can extend the example to include actions and queries that are performed by procedural attachments – utilizing the *situated* extension of (courteous) logic programs.

#### **Example 4 (Ordering Lead Time Management, in SCLP)**

Next, we extend Example 2 to include actions and queries that are performed by procedural attachments. We add the following rules, effector statements, and sensor statements:

- (Rule D:) “Accept a request to modify an order if it is received before the lead time.”
- (Rule E:) “Deny a request to modify an order if it is not received after the lead time.”
- (Action Rule F:) “Update the orders database if a request to modify an order is accepted.”
- (Action Rule G:) “Remind the customer of the lead time policy if a request to modify an order is denied.”
- (Action Rule H:) “Notify the customer if a request to modify an order is accepted.”
- (Action Rule I:) “Notify the customer if a request to modify an order is denied.”

$\langle d \rangle$  *acceptOrderModificationRequest(?Request)*  
 $\leftarrow$  *modifies(?Request, ?Order) \wedge*  
*orderModificationNotice(?Order, ?LeadTime) \wedge*  
*deliveryDate(?Order, ?Day1) \wedge*  
*receiptDate(?Request, ?Day2) \wedge*  
*subtractDate(?Day1, ?LeadTime, ?Day3) \wedge*  
*lessThanOrEqual(?Day2, ?Day3).*

$\langle e \rangle$  *denyOrderModificationRequest(?Request)*  
 $\leftarrow$  *modifies(?Request, ?Order) \wedge*  
*orderModificationNotice(?Order, ?LeadTime) \wedge*  
*deliveryDate(?Order, ?Day1) \wedge*

- $$\begin{aligned}
& receiptDate(?Request, ?Day2) \wedge \\
& subtractDate(?Day1, ?LeadTime, ?Day3) \wedge \\
& greaterThan(?Day2, ?Day3). \\
\langle f \rangle \quad & shouldUpdateOrderDb(?Request) \\
& \leftarrow acceptOrderModificationRequest(?Request). \\
\langle g \rangle \quad & shouldRemindCustomerOfPolicy(?Request) \\
& \leftarrow denyOrderModificationRequest(?Request). \\
\langle h \rangle \quad & shouldInformCustomer(?Request, accepted) \\
& \leftarrow acceptOrderModificationRequest(?Request). \\
\langle i \rangle \quad & shouldInformCustomer(?Request, denied) \\
& \leftarrow denyOrderModificationRequest(?Request).
\end{aligned}$$

The following *effector statements* each associate a pure-belief predicate, e.g., *shouldInformCustomer*, with an external procedure (here, a Java method), e.g., *orderMgmt.request.mods.ack*. During rule inferencing (more precisely, during rule execution), when a conclusion is drawn about the predicate, e.g., *shouldInformCustomer(request1049, accepted)*, then the external procedure is invoked as a side-effectful action, e.g., the method *orderMgmt.request.mods.ack* is called with its parameters instantiated to *(request1049, accepted)*.

- $$\begin{aligned}
& shouldInformCustomer(?X, ?Y) \\
& \quad :: e :: orderMgmt.request.mods.ack(?X, ?Y). \\
& shouldUpdateOrderDB(?Request) \\
& \quad :: e :: orderMgmt.request.mods.updOrderDB(?Request). \\
& shouldRemindOfPolicy(?Request) \\
& \quad :: e :: orderMgmt.request.mods.remind(?Request).
\end{aligned}$$

The following *sensor statements* each associate a pure-belief predicate, e.g., *receiptDate*, with an external procedure (here a Java method), e.g., *orderMgmt.request.getReceiptDate*. During rule inferencing/execution, when a rule antecedent condition (i.e., a literal in the rule's "if" part) is tested,

e.g., *receiptDate(?Request, ?Day2)*, the external procedure is queried to provide information about that condition’s truth (more precisely, for its answer bindings). Some sensor statements, e.g., for the predicate *lessThanOrEqual*, correspond to what in Prolog (or many other commercial rule systems) are “built-ins”, utility procedures provided as a standard package with the rule system rather than specified by a particular individual user/application.

*receiptDate(?X, ?Y)*

*:: s :: orderMgmt.request.getReceiptDate(?X, ?Y).*

*deliveryDate(?X, ?Y)*

*:: s :: orderMgmt.order.getDelivDate(?X, ?Y).*

*subtractDate(?X, ?Y, ?Z)*

*:: s :: utils.date.subtract(?X, ?Y, ?Z).*

*lessThanOrEqual(?X, ?Y)*

*:: s :: utils.arith.lte(?X, ?Y).*

*greaterThan(?X, ?Y)*

*:: s :: utils.arith.gt(?X, ?Y).*

## 4 RuleML Syntax: DTD for SCLP

Next, we specify the XML syntax for SCLP in RuleML (the version current as of this paper’s version date). In so doing, we extend the syntax of RuleML syntax to SCLP – from the previous existing Ordinary Logic Programs (OLP) case of the syntax. This is a novel contribution by us. This syntax specification takes the form of an XML Document Type Definition (DTD).

The extant DTD version of the XML syntax is currently being extended to versions using XML Schema and RDF.

The following is the current RuleML DTD for SCLP; this extends RuleML V0.8 syntax for OLP.

<!--

DOCUMENTATION:

rulebase is short for rule knowledge base, i.e., rule set

rbaselab is short for rulebase label

rlab is short for rule label

rel is short for relation or predicate (constant)

ind is short for individual (constant)

opr is short for relation operator

andb is short for and expression in body

orb is short for or expression in body

andh is short for and expression in head

ando is short for and expression in the opposers part  
of a mutex statement

ctor is short for constructor (logical function)

opc is short for constructor operator

bool is short for boolean

bind is short for binding status, i.e., bmode value

lit is short for literal

clit is short for classical literal

fclit is short for failure classical literal

cneg is short for classical negation

fneg is short for failure negation, i.e., negation as failure,  
a.k.a. weak negation or default negation

oppo is short for opposers

mgiv is short for mutex\_given part of mutex

suffixes for and, or: h is short for head, b is short for body,

o is short for opposers

aproc is short for attached procedure

meth is short for method (name)

clas is short for class (name)

sens is short for sensor link statement

effe is short for effector link statement

aproc is short for attached procedure (description)

jproc is short for java procedure (description)

uproc is short for universal-protocol/un-particularized  
procedure (description)

modli is short for bmode list, i.e., binding signature list

bmode is short for binding mode

-->

<!-- SCLP RuleML DTD with URI aspects,

Monolith version of 2001-12

by Benjamin Grosf -->

<!ENTITY % URI "CDATA">

<!ENTITY % bool "yes|no">

<!ELEMENT rulebase (

  (\_rbaselab, (imp | fact | mutex | sens | effe)\*)

  | ((imp | fact | mutex | sens | effe)+, \_rbaselab?) )>

<!ATTLIST rulebase direction

  (forward | backward | bidirectional)

  "bidirectional">

```

<!ELEMENT _rbaselab (ind | cterm)>
<!-- in this version,
      we do NOT require imp to have a (non-empty) body,
      rather an imp is a permutation of
      the UNORDERED {_rlab?,_head,_body?}: -->
<!ELEMENT imp ( (_head, ((_body,_rlab?) | (_rlab,_body?))? )
                | (_body, ((_head,_rlab?) | (_rlab,_head)))
                | (_rlab,((_head,_body?) | (_body,_head))) )>
<!-- ff. is the alternative version,
      closer to V0.8 datalog-monolith,
      that DOES require imp to have a body,
      i.e., be a permutation of
      the UNORDERED {_rlab?,_head,_body?}: -->
<!--
<!ELEMENT imp ( (_head,((_body,_rlab?) | (_rlab,_body)))
                | (_body, ((_head,_rlab?) | (_rlab,_head)))
                | (_rlab,((_head,_body?) | (_body,_head))) )>
-->
<!ELEMENT _rlab (ind | cterm) >
<!ELEMENT fact ( (_rlab,_head) | (_head,_rlab?) )>
<!ELEMENT _head (clit | atom | andh)>
<!ELEMENT _body (fclit | atom | clit | flit | andb | orb | and)>
<!ELEMENT andb ((fclit | atom | clit | flit | andb | orb)*)>
<!ELEMENT orb ((fclit | atom | clit | flit | andb | orb),
               (fclit | atom | clit | flit | andb | orb)+)>
<!ELEMENT andh ((clit | atom | andh), (clit | atom | andh)+)>
<!ELEMENT and ((atom | and)*)>
<!ELEMENT clit ((_opr, (ind | var | cterm)*)

```

```

        | ((ind | var | cterm)+, _opr))>
<!ATTLIST clit cneg (%bool;) #IMPLIED>
<!ELEMENT fclit ((_opr, (ind | var | cterm)*
                | ((ind | var | cterm)+, _opr))>
<!ATTLIST fclit cneg (%bool;) #IMPLIED>
<!ATTLIST fclit fneg (%bool;) #IMPLIED>
<!ELEMENT flit ((_opr, (ind | var | cterm)*
                | ((ind | var | cterm)+, _opr))>
<!ATTLIST flit fneg (%bool;) #IMPLIED>
<!ELEMENT atom ((_opr, (ind | var | cterm)*
                | ((ind | var | cterm)+, _opr))>
<!ELEMENT _opr (rel)>
<!ELEMENT rel (#PCDATA)>
<!ATTLIST rel href %URI; #IMPLIED>
<!ELEMENT var (#PCDATA)>
<!ELEMENT ind (#PCDATA)>
<!ATTLIST ind href %URI; #IMPLIED>
<!ELEMENT cterm ((_opc, (ind | var | cterm)*
                | ((ind | var | cterm)+, _opc))>
<!ELEMENT _opc (ctor)>
<!ELEMENT ctor (#PCDATA)>
<!ATTLIST ctor href %URI; #IMPLIED>

<!-- syntax for courteous and situated follows --->

<!ELEMENT mutex ((_oppo, _mgiv?) | (_mgiv, _oppo))>
<!ELEMENT _oppo (ando)>
<!ELEMENT _mgiv (fclit | andb | orb)>

```

```

<!ELEMENT ando (clit, clit)>

<!ENTITY % bind "bound|free">
<!ELEMENT sens ((_opr, ((_aproc, _modli?) | (_modli,_aproc)))
                | (_aproc, ((_opr,_modli?) | (_modli,_opr)))
                | (_modli,((_aproc,_opr) | (_opr,_aproc)))) >>
<!ELEMENT effe ((_opr, _aproc) | (_aproc, _opr))>
<!ELEMENT _aproc (jproc | uproc)>
<!ELEMENT uproc (#PCDATA)>
<!ATTLIST uproc href %URI; #IMPLIED>
<!ELEMENT jproc ((clas, ((meth, path?) | (path, meth)))
                 | (meth, ((clas, path?) | (path, clas)))
                 | (path, ((meth, clas) | (clas, meth))))>
<!ELEMENT path (#PCDATA)>
<!ATTLIST path href %URI; #IMPLIED>
<!ELEMENT clas (#PCDATA)>
<!ATTLIST clas href %URI; #IMPLIED>
<!ELEMENT meth (#PCDATA)>
<!ATTLIST meth href %URI; #IMPLIED>
<!ELEMENT _modli ((bmode)*)>
<!ELEMENT bmode EMPTY>
<!ATTLIST bmode val (%bind;) "free">

```

## 5 Example of a SCLP in RuleML

Next we give a basic example of a SCLP in RuleML. RuleML inherits the verbose quality of XML. We thus show only the first Ordering-Lead-Time rule

in detail, for the sake of brevity. More details on syntax and examples are available on the author's website.

```
<?xml version="1.0" ...>
<!DOCTYPE rulebase SYSTEM ...>

<rulebase>

<imp>
  <_head>
    <atom>
      <_opr><rel>orderModificationNotice</rel></_opr>
      <var>Order</var>
      <ind>days14</ind>
    </atom>
  </_head>
  <_body>
    <andb>
      <atom>
        <_opr><rel>preferredCustomerOf</rel></_opr>
        <var>Buyer</var>
        <var>Seller</var>
      </atom>
      <atom>
        <_opr><rel>purchaseOrder</rel></_opr>
        <var>Order</var>
        <var>Buyer</var>
        <var>Seller</var>
      </atom>
    </andb>
  </_body>
</imp>
</rulebase>
```

```
        </atom>
    </andb>
</_body>
</imp>

...

</rulebase>
```

## 6 SWEET Prototype of SCLP in RuleML

We have for the first time designed and prototyped how to perform inferencing and translation for situated courteous logic programs (SCLP) in RuleML. The prototype system is a set of tools called SWEET, short for “Semantic Web Enabling Technology”.

We have also for the first time given a specification for the syntax of RuleML for the case of SCLP. To do this, we extended the previously existing Ordinary Logic Programs (OLP) case of the syntax for RuleML – which lacks the prioritized conflict handling (courteous) and procedural attachment (situated) features.

Our tools are implemented using XSLT<sup>17</sup> (a tool for transforming from one XML format/syntax into another) and Java, and make use of IBM Common-Rules as a middle component. The prototype tools implement both forward and backward inferencing for SCLP RuleML. For backward inferencing, we make use of the XSB<sup>18</sup> logic programming system. The inferencing tools take SCLP RuleML premises as input and return SCLP RuleML conclusions as out-

---

<sup>17</sup>see, e.g., <http://www.w3.org/TR/xslt>

<sup>18</sup><http://www.sunysb.edu/~sbprolog>

put. The tools also implement translation of RuleML to/from several other rule system representations, including Prolog, Knowledge Interchange Format (KIF), and another XML format. We are using the tools to run several examples taken from e-business application domains, including supply chain leadtime, e-bookstore pricing, and creditworthiness.

The first version of SWEET was demonstrated at the Workshop on Information Technology and Systems (WITS '01) on Dec. 16, 2001 as part of that Workshop's refereed system demonstration program. The version of SWEET demonstrated there implemented fully general Courteous LP, but not yet most Situated features.

## 7 Current, Related, and Future Work

Further additions to SWEET's functionality are currently being developed by our research group.

A Web-runnable version, and a free trial downloadable version, of SWEET are planned for the near future.

Also in current work, we are exploring how to provide a Rule mechanism to (other) emerging W3C<sup>19</sup> standards, especially: standards for the Semantic Web, notably its ontologies<sup>20</sup> aspect (WebOnt) and RDF; XML Query for XML database querying; and P3P's APPEL rule language for user privacy policies. Each of these areas has an associated significant body and community of related fundamental research.

One direction for future work is pilot applications in e-business, including: contracting and negotiation, e.g., where rules are useful to represent product/service/deal descriptions; search, selection, and composition of Web Services; and security & privacy, e.g., where rules are useful to represent au-

---

<sup>19</sup><http://www.w3.org>

<sup>20</sup>structured vocabularies specified using logic-based KR

thorization policies. Here we would welcome a community of researchers and practitioners trying out the RuleML approach, developing requirements, and providing a feedback loop to further improvements and extensions.

## Acknowledgements

Harold Boley of DFKI (Germany) and Said Tabet of Nisus Inc. (USA) are my close collaborators in leading the RuleML Initiative. Hoi Chan of IBM T.J. Watson Research Center contributed much to the design details of Business Rules Markup Language. Youssef Kabbaj, student at MIT, contributed much to the RuleML prototype's implementation.

## References

- [1] Chitta Baral and Michael Gelfond. Logic programming and knowledge representation. *Journal of Logic Programming*, 19,20:73–148, 1994. Includes extensive review of literature.
- [2] Benjamin N. Grosf. Building Commercial Agents: An IBM Research Perspective (Invited Talk). In *Proceedings of the Second International Conference and Exhibition on Practical Applications of Intelligent Agents and Multi-Agent Technology (PAAM97)*, P.O. Box 137, Blackpool, Lancashire, FY2 9UN, UK. <http://www.demon.co.uk./ar/PAAM97>, April 1997. Practical Application Company Ltd. Held London, UK. Also available as IBM Research Report RC 20835 at World Wide Web <http://www.research.ibm.com> .
- [3] Benjamin N. Grosf. Courteous logic programs: Prioritized conflict handling for rules. Technical report, IBM T.J. Watson Research Center, <http://www.research.ibm.com> , search for Research Reports; P.O. Box 704, Yorktown Heights, NY 10598, Dec. 1997. IBM Research Report RC 20836. This is an extended version of [4].

- [4] Benjamin N. Grosf. Prioritized Conflict Handling for Logic Programs. In Jan Maluszynski, editor, *Logic Programming: Proceedings of the International Symposium (ILPS-97)*, pages 197–211, Cambridge, MA, USA, 1997. MIT Press. Held Port Jefferson, NY, USA, Oct. 12-17, 1997. <http://www.ida.liu.se/~ilps97>. Extended version available as IBM Research Report RC 20836 at <http://www.research.ibm.com> .
- [5] Benjamin N. Grosf. Compiling Prioritized Default Rules Into Ordinary Logic Programs. Technical report, IBM T.J. Watson Research Center, <http://www.research.ibm.com> , search for Research Reports; P.O. Box 704, Yorktown Heights, NY 10598. USA, May 1999. IBM Research Report RC 21472.
- [6] Benjamin N. Grosf. A Courteous Compiler from Generalized Courteous Logic Programs To Ordinary Logic Programs (Preliminary Report). Technical report, IBM T.J. Watson Research Center, <http://www.research.ibm.com/people/g/grosf/papers.html> ; P.O. Box 704, Yorktown Heights, NY 10598. USA, July 1999. This is a supplementary followon to IBM Research Report RC 21472. Revised version forthcoming as another IBM Research Report. Included as part of documentation in the IBM CommonRules 1.0 alpha prototype Web release of July 30, 1999 at <http://alphaworks.ibm.com> .
- [7] Benjamin N. Grosf, Yannis Labrou, and Hoi Y. Chan. A Declarative Approach to Business Rules in Contracts: Courteous Logic Programs in XML. In Michael P. Wellman, editor, *Proceedings of the 1st ACM Conference on Electronic Commerce (EC-99)*. ACM Press, 1999. Held in Denver, CO.