To appear in *Electronic Commerce Research and Applications*,

Accepted Journal Paper, revised version of Sept. 29, 2003

Representing E-Commerce Rules Via Situated Courteous Logic Programs in RuleML

Benjamin N. Grosof

Massachusetts Institute of Technology
Sloan School of Management
50 Memorial Drive
Cambridge, MA 02142, USA
bgrosof@mit.edu
http://ebusiness.mit.edu/bgrosof

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 co-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 SweetRules, (Semantic WEb Enabling Technology – Rules component), 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 in supply chain management. [1]

Key words: e-business, e-commerce, rules, logic programs, Semantic Web, RuleML, knowledge representation, non-monotonic reasoning, interoperability, industry standards, e-contracting, XML, supply chain management

[1] This paper is extended and updated from an earlier workshop paper [7].

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 co-lead. The Rule Markup Language Initiative [2] is part of the R&D community's effort to develop the *Semantic Web* [3]. The Semantic Web is a vision of moving the Web to support program-to-program communication in which data has high-level semantics that is shared by both communicating parties (programs). RuleML is based on a fundamental rule KR, declarative *logic programs* (LP). In previous work [11,2,4,3,5,6] we have expressively extended ordinary LP (OLP) [4] 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 *SweetRules*. SweetRules is part of our larger toolset *SWEET*, short for "Semantic WEb Enabling Technology". SweetRules enables communication and inferencing of e-business rules represented in RuleML. We have a running prototype of SweetRules which was first demonstrated at the Workshop on Information Technology and Systems (WITS '01) in Dec. 2001, as part of the refereed software systems demonstration program there [5]. That demonstration complemented an earlier, briefer workshop version of this paper [7].

We illustrate SCLP RuleML by giving example rulesets from the realm of e-commerce business policies in supply chain management, specifically about managing ordering lead time of purchase requests.

---

[2] see http://www.ruleml.org and the author's home page (http://ebusiness.mit.edu/bgrosof)

[3] see the overview of the Semantic Web Activity on the World Wide Web Consortium's website http://wwww.w3.org

[4] sometimes called "general" LP or "normal" LP; see [1] for a helpful review

[5] http://www.busi.mun.ca/parsons/wits2001/

2 RuleML and expressively extended Logic Programs

We are leading, with Harold Boley and Said Tabet, an early-phase standards effort on a markup language for exchange of rules in XML, called RuleML (Rule Markup Language) [6]. The goal of this effort is eventual adoption as a Web standard, e.g., via the World Wide Web Consortium (W3C) [7] within its new Semantic Web Activity. [8] 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. As we first observed [11], these four families of rule systems all have common core abstraction: declarative logic programs (LP) [9]. "Declarative" here means in the sense of KR theory. [10] Note that this supports both backward inferencing and forward inferencing.

RuleML is actually a family (lattice) of rule KR expressive classes [11]: 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). 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, although there are many details to work out. Crucial, however, 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 – and the rule-based applications using them – that are exchanging rules.

The kernel representation in RuleML is: Horn declarative logic programs. Ex-

[6] http://www.ruleml.org and http://ebusiness.mit.edu/bgrosof/#RuleML
[7] http://www.w3.org
[8] http://www.w3.org/2001/sw
[9] [1] provides a helpful review of declarative LP
[10] 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").
[11] "Class" here means an expressive subset of a logical language.

tensions to this representation are defined for several additional expressive features:

- negation-as-failure (NAF). Prolog, for example, includes NAF.
- prioritized conflict handling: cf. *courteous* logic programs [11].
- disciplined procedural attachments for queries and actions: e.g., cf. *situated* logic programs [2]

and other features as well. The feature of limited classical negation, a.k.a. "*strong*" negation, is included as part of the courteous extension of LP. In addition, RuleML defines some useful expressive restrictions (e.g., Datalog, facts-only, binary-relations-only), not only expressive generalizations. The extension of Horn LP to include NAF is called "*ordinary*" LP (OLP), a.k.a. "normal" LP.

In Jan. 2001, we (in collaboration with Harold Boley and Said Tabet) 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 [12] held Feb. 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. The current stable version is V0.8.

RuleML largely grows out of the design approach and design criteria of Business Rules Markup Language (BRML) which was developed in our previous work during 1997–2000 at IBM Research and which is implemented in IBM CommonRules [13] available under free trial license from IBM alphaWorks. The first version of IBM CommonRules, with BRML, was publicly released in July 1999. The design approach and design criteria of CommonRules and BRML are described in [11,10], 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.

3 Advantages of Situated Courteous Logic Programs KR

SCLP and BRML were developed in large part to surmount the limitations of the first major attempt at a standardizing a declarative KR for knowledge interchange: Knowledge Interchange Format (KIF) [14]. KIF was developed in

[12] a large convocation of most of its face-to-face standards working group meetings
[13] http://www.research.ibm.com/rules and http://alphaworks.ibm.com
[14] http://logic.stanford.edu/kif and http://www.cs.umbc.edu/kif/

the early 1990's as a system for researchers – rather than commercial applications – to exchange logical-form knowledge. It thus preceded the Web. KIF's KR is essentially classical logic. A newer early phase standards effort called (Simple) Common Logic [15] is the close succcessor to KIF. Launched in about fall 2000, Common Logic is aiming for status as an ISO standards effort. Like KIF, Common Logic is based on the underlying KR of classical logic, primarily First Order Logic (FOL). Classical logic is the basis for most of mathematics, and FOL is familiar and popular among mathematicians.

The (declarative) logic programs KR expressively overlaps with FOL: the Horn subset of LP is also a subset of FOL. More generally, however, full LP includes expressive features (e.g., negation-as-failure and procedural attachments) that FOL cannot express. And, vice versa, full FOL includes expressive features (e.g., existentials and disjunctions) that LP cannot express.

FOL (beyond LP) has not become widely used for deployed commercial applications. Likewise, KIF and Common Logic have not either – no need to interchange lots of FOL knowledge if there are not lots of applications using FOL knowledge. In our view, the lack of wide commercial deployment of FOL (beyond LP) is because it fundamentally lacks the ability to express non-monotonicity or procedural attachments, and partly also because of its (computational) intractability.

In particular, FOL has two major expressive limitations that prevent it from representing e-business rules of the kind typically 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. [16] Yet procedural attachments and logical non-monotonicity are heavily used in CCI rule systems and their applications. Negation-as-failure is the usual and most important form of negation found in all the CCI rule systems and their applications.

Example kinds of non-monotonic prioritized conflict handling heavily used in CCI rule systems include:

- priority between rules in Prolog based on static rule sequence;
- priorities (in general, dynamically-computed) among rules in production rule and ECA rule systems;
- inheritance with exceptions; and
- updating in databases (where more recent assertions override previous ones).

[15] http://cl.tamu.edu

[16] A particular KR is said to be logically monotonic when it has the property that adding premises never results in retracting (sanctioned) conclusions.

In contrast to FOL, situated courteous logic programs provide expressive features both for procedural attachments – in actions and queries – and for non-monotonicity – negation-as-failure and prioritized conflict handling. 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, and logical functions are disallowed; classical logic. By contrast, classical logic inferencing is NP-hard for this case.

4 Webizing KR

As we discussed earlier, the SCLP KR is supported in BRML, the XML "interlingua" syntax of IBM CommonRules. RuleML in its current V0.8 version advances upon its BRML predecessor in several significant respects, however. One respect is that it defines a family of DTD's rather than just one. More deeply, however, these advances largely revolve around "Webizing" the KR, notably:

- labels (names) for rules/rulebases; in particular, this aids import/export. [17]
- URI's [18] for logical vocabulary and knowledge subsets: predicates, functions, rules, rulebases, labels for rules and rulebases.
- headers: meta-data describes the XML document's expressive class.
- procedural attachments using Web protocols/services; queries or actions via CGI, SOAP, and/or other Web Services mechanisms.

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 Web standards, especially from W3C and Oasis [19], e.g., for namespaces, for Web Services, etc.
- share mechanisms with other emerging or existing standards for KR and ontologies [20] on the Web, including especially W3C's RDF [21] and OWL [22].
- have rules (or other knowledge) permit and utilize (URI) references to ontologies (knowledge bases), e.g., to have a rulebase refer to predicates or individuals previously defined in an OWL ontology [12].

[17] BRML did previously include labels for rules.
[18] Uniform Resource Identifiers, a generalization of Uniform Resource Locators (URL's), a W3C standard
[19] http://www.oasis-open.org
[20] structured vocabularies specified using logic-based KR
[21] http://www.w3.org/RDF
[22] http://www.w3.org/2001/sw/WebOnt

- more generally, use ontologies for rules, and rules for ontologies, e.g., with semantics of their combination defined using the KR of Description Logic Programs [9].
- have a knowledge base include some self-description in terms of an OWL ("meta-")ontology, e.g., to have a RuleML rulebase specify which expressive features it employs and expressive restrictions it obeys.

Additionally, RuleML V0.8 has some further steps of Webizing rule KR, including:

- Slots. RuleML V0.8 supports object-oriented style slots (a.k.a. "roles") to specify arguments of a predicate (or of a logical function), in addition to Prolog-style sequentially ordered positions for arguments. Such slots of a predicate (or function) are similar to the named members of a Java/C++ class.
- Unorderedness of the syntactic data model. This facilitates an (alternative) RDF syntax for RuleML: by having RuleML's abstract syntax (and XML syntax) avoid reliance on ordered-ness of child elements within any element (there are already some partial first drafts of such an alternative RDF syntax). The only exception in RuleML V0.8 XML syntax is to permit orderedness where orderedness is explicitly desired: when representing ordered tuples, in the "tup" (and "bmtup") elements.

5 Example: Ordering Lead Time

In this section, we illustrate how to use SCLP to represent e-commerce rules with a long example. As we go, we describe some more of the SCLP KR's technical aspects and advantages, including the advantages of the courteous feature.

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. This kind of information is desirable and important to communicate between the buyer and seller — preferably, automatically, i.e., in e-commerce. The seller want its buyers to know this, to clarify its negotiating or competitive position, to smooth operations and collaboration, and to avoid misunderstandings and problems. Furthermore, the business rules about ordering lead time often need to be shared among multiple applications within the buyer organization, and among multiple applications within the seller organization. Ordering lead time affects not only planning and scheduling both for the buyer and for the seller. It also affects accounting, cash flow, inventory management, operations, partnering strategy, etc. It ripples down to bidding/negotiation and

order management with the seller's lower-tier suppliers and, likewise, ripples up to higher-tier buyers. Indeed, over the last two decades, much of the focus of supply chain management (SCM) has been on lead time. Reducing the variance, as well as the mean, of lead time is at the heart of creating and exploiting opportunities to reduce inventory and cycle times in the supply chain. Smaller inventories or cycle times improve productivity by reducing working capital, warehousing, obsolescence and wasteage, as well as buyer opportunity costs.

Example 1 (Ordering Lead Time)

An example of a parts supplier vendor's lead time policy, specified in natural language, 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 1:) "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. Priority Rule 1 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)

Example 1 above can be straightforwardly represented in CLP as the following rulebase $RB1$:

$\langle a \rangle \quad orderModificationNotice(?Request, days14)$
$\quad \leftarrow \quad preferredCustomerOf(?Buyer, ?Seller) \wedge$
$\quad\quad purchaseRequest(?Request, ?Buyer, ?Seller);$
$\langle b \rangle \quad orderModificationNotice(?Request, days30)$
$\quad \leftarrow \quad minorPart(?Request) \wedge$
$\quad\quad purchaseRequest(?Request, ?Buyer, ?Seller);$

$\langle c \rangle \quad orderModificationNotice(?Request, days2)$
$\leftarrow \quad preferredCustomerOf(?Buyer, ?Seller) \wedge$
$orderModificationType(?Request, reduce) \wedge$
$orderItemIsInBacklog(?Request) \wedge$
$purchaseRequest(?Request, ?Buyer, ?Seller);$
(*End of Example*)

Here and in the rest of this paper, as notation for SCLP, we are employing an extended form of the Prolog-like notation for declarative LP that is typically used in the LP literature. The implication symbol "$\leftarrow$" can be read as "if". The conjunction symbol "$\wedge$" can be read as "and". The consequent part of the rule (left of the "$\leftarrow$") is also called the *head* of the rule. The antecedent part of the rule (right of the "$\leftarrow$") is also called the *body* of the rule. The delimiter ";" ends a rule statement. The prefix "?" indicates a logical variable. A fact is a special case of a rule; its body (i.e., antecedent) is empty, and the "$\leftarrow$" may be omitted. (An empty body can be viewed as logically true.) "$< \ldots >$" encloses a rule label. The label is optional. $\sim$ stands for negation-as-failure. (Example 3 below illustrates the use of $\sim$).

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 higher priority than its second argument. *overrides* is otherwise an ordinary predicate, e.g., *overrides* facts can be inferred via rules. In general, the prioritization ordering is a *partial* ordering rather than a total ordering. This aids flexibility for specification and merging of rulesets.

In order to specify that c has higher priority than a, we add the following rule to our example rulebase. This rule is a fact:
$\langle pri1 \rangle \quad overrides(c, a);$

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 it is a contradiction/inconsistency for a particular pair of literals (known as the "*opposers*") to be inferred, given another particular condition. 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.

We add to our example rulebase the following mutex, which specifies that it is a contradiction to conclude two different values of the ordering-lead-time for the same order:
$\bot \quad \leftarrow \quad orderModificationNotice(?Request, ?X) \wedge$
$orderModificationNotice(?Request, ?Y)$
$\mid \; notEquals(?X, ?Y);$

Here, we have introduced some additional SCLP notation. The "$\perp''$ stands for logical contradiction. The " | " appearing to the right of the first, opposers part of the mutex can be read as "given", and is followed by a condition expression that is similar to a rule body. Note that in the Courteous semantics aspect of SCLP, the scope of conflict, and the associated enforcement of consistency, is defined in terms of the two opposer literals, for each mutex. [23]

The ordering lead time policy rules are mainly interesting insofar as they enable one to infer ordering lead times for particular ordering situations. Next, we continue our example. Let the seller be a company named "ElvesCo". Suppose one adds to the rulebase some facts about a particular ordering situation:

$\langle sit11 \rangle \quad preferredCustomerOf(EagerCo, ElvesCo);$

$\langle sit12 \rangle \quad purchaseRequest(pu34, EagerCo, ElvesCo);$

The desired conclusion for this ordering situation is that the ordering lead time is 14 days, since only Rule A applies. As desired, under the semantics of the SCLP KR, the CLP rulebase $RB1$ indeed entails this:

$RB1 \models orderModificationNotice(pu34, days14);$

(Notation: $\models$ stands for the entailment relationship.)

More interestingly, suppose one adds to the rulebase some more facts about another particular ordering situation in which both Rule C and Rule A apply:

$\langle sit21 \rangle \quad preferredCustomerOf(BeaverCo, ElvesCo);$

$\langle sit22 \rangle \quad purchaseRequest(pu56, BeaverCo, ElvesCo);$

$\langle sit23 \rangle \quad orderModificationType(pu56, reduce);$

$\langle sit24 \rangle \quad orderItemIsInBacklog(pu56);$

The desired conclusion for this second ordering situation is that the ordering lead time is 2 days, since although both Rule A and Rule C apply and conflict with each other, that conflict is resolved by the priority. As desired, the semantics of the SCLP KR indeed entails this:

$RB1 \models orderModificationNotice(pu56, days2);$

Also as desired, the rulebase does *not* entail any other values (e.g., $days14$) for $pu56$'s ordering lead time. [24]

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

Next, we give a direct Ordinary LP representation of Example 1 (i.e., without the courteous expressive features) This rulebase, let's call it $RB2$, essentially behaves equivalently semantically to the Courteous LP version ($RB1$ in Example 2). That is, it entails the same conclusions about $orderModificationNotice$,

[23] The distinction between the (exactly two) literals appearing in the *opposer* part of a mutex versus literals appearing in the *given* part of that mutex is, therefore, not just syntactic sugar; rather, that distinction has semantic significance. The scope of conflict, and associated enforcement of consistency, is *not* specified in terms of n-ary groups of literals for n>2.

[24] A simpler version of Example 2, lacking the particular situation facts, appeared in [11].

for various ordering situations.

$\langle a1 \rangle$ $orderModificationNotice(?Request, days14)$
$\leftarrow$ $preferredCustomerOf(?Buyer, ?Seller) \wedge$
$purchaseRequest(?Request, ?Buyer, ?Seller) \wedge$
$\sim minorPart(?Request) \wedge$
$\sim orderItemIsInBacklog(?Request)$;

$\langle a2 \rangle$ $orderModificationNotice(?Request, days14)$
$\leftarrow$ $preferredCustomerOf(?Buyer, ?Seller) \wedge$
$purchaseRequest(?Request, ?Buyer, ?Seller) \wedge$
$\sim minorPart(?Request) \wedge$
$\sim orderModificationType(?Request, reduce)$;

$\langle b \rangle$ $orderModificationNotice(?Request, days30)$
$\leftarrow$ $minorPart(?Request) \wedge$
$purchaseRequest(?Request, ?Buyer, ?Seller) \wedge$
$\sim preferredCustomerOf(?Buyer, ?Seller)$;

$\langle c \rangle$ $orderModificationNotice(?Request, days2)$
$\leftarrow$ $preferredCustomerOf(?Buyer, ?Seller) \wedge$
$orderModificationType(?Request, reduce) \wedge$
$orderItemIsInBacklog(?Request) \wedge$
$purchaseRequest(?Request, ?Buyer, ?Seller) \wedge$
$\sim minorPart(?Request)$;

$notEquals(days14, days2)$;
$notEquals(days14, days30)$;
$notEquals(days2, days30)$;
$\langle sit11 \rangle$ $preferredCustomerOf(EagerCo, ElvesCo)$;
$\langle sit12 \rangle$ $purchaseRequest(pu34, EagerCo, ElvesCo)$;
$\langle sit21 \rangle$ $preferredCustomerOf(BeaverCo, ElvesCo)$;
$\langle sit22 \rangle$ $purchaseRequest(pu56, BeaverCo, ElvesCo)$;
$\langle sit23 \rangle$ $orderModificationType(pu56, reduce)$;
$\langle sit24 \rangle$ $orderItemIsInBacklog(pu56)$;

(Notation: recall that $\sim$ stands for negation-as-failure.)
Note that typically in practical implemented rule systems, the *notEquals* facts above would be provided in a "built-in" manner rather than having to be explicitly stated as facts. (These facts are needed because *notEquals* appears in the given part of the mutex about *orderModificationNotice*.) Later in Example 4, we will show how to use the Situated feature of SCLP to specify that in a simple manner.

As desired, $RB2$ entails that the lead times for ordering situations $pu34$ and $pu56$ are 14 days and 2 days, respectively:
$RB1 \models orderModificationNotice(pu34, days14)$;
$RB1 \models orderModificationNotice(pu56, days2)$;

We can state the $RB2$'s lead time rules ($a1$, $a2$, b, c) in natural language as

follows:
• (Rule A1:) "14 days lead time if: the buyer is a preferred customer, *and the item is NOT a minor part, and the item is NOT backlogged*."
• (Rule A2:) "14 days lead time if: the buyer is a preferred customer, *and the item is NOT a minor part, and the order is NOT a modification to reduce the quantity of the item*."
• (Rule B:) "30 days lead time if: the ordered item is a minor part, *and the buyer is NOT a qualified customer*."
• (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, *and the ordered item is NOT a minor part*."

Comparing $RB2$ to the Courteous LP representation's rulebase $RB1$ from Example 2, one sees that $RB2$ not only omits the priority (*overrides*) fact and the mutex statement, which we expect since $RB2$ is employing the Ordinary LP KR rather than the more expressive Courteous LP KR which features prioritization and mutex's. One sees also that $RB2$ *modifies* the lead time rules. It *adds negated (∼'d) "interaction" conditions to the bodies of the lead time rules, so as to avoid overlap cases where the rules might conflict*. Above in the natural language statement of $RB2$'s lead time rules, the differences from the natural language statement of $RB1$'s lead time rules are italicized so as to highlight the comparison. These NAF interaction conditions in effect handle conflict appropriately in regard to priorities and guarantee consistency w.r.t. the desired uniqueness of lead time. If instead these negation-as-failure interaction conditions were omitted then undesired extra conclusions would be drawn — e.g., that the request $pu56$ *also* has lead time 14 days!

Furthermore, in this direct OLP style of representation, adding a new rule (e.g., adding Rule C to a rulebase containing just Rule A and Rule B) requires modifying the other rules to add additional such interaction conditions. This creates maintenance and scaleability problems in specification over the lifecycle of the rulebase and the applications that use it. This need for modification (when updating in an OLP representation) is typical of conflicting rule-sets and underscores the advantage of the prioritized conflict handling expressive feature for modularity and ease of modification. Observe that Example 1 did not use "not", "but", "unless", or any other close natural language correspondent of negation. And our CLP specification of it in Example 2 did not employ negation. The form of modifications needed, in general, when updating or conflict-handling in an OLP representation, moreover, go beyond simply and'ing (i.e., conjoining) new negated body atoms in individual rules. Rule A in $RB1$ actually became in $RB2$ *two* rules A1 and A2. [25]

[25] This was essentially due to the need to represent (in the body of modified Rule A) the negation of a conjunction of atoms, which is equivalent to the *disjunction* of

(*End of Example*)

The example above thus illustrates how using Ordinary LP directly to specify rulesets that need conflict handling or priorities, is clumsier and lower-level than using Courteous LP, especially in regard to updating and avoiding overlap cases where the rules might conflict.

Above, we called Example 3 a "direct" OLP version. Actually, as we have shown in previous work [11,5,6], every (Situated) Courteous LP is expressively reducible to a semantically equivalent (Situated) Ordinary LP, via use of a transform called the Courteous Compiler. IBM CommonRules and SweetRules make use of a Courteous Compiler component, for example; CommonRules provides one as part of its toolset. The Courteous Compiler step is tractable computationally: worst-case $O(n^3)$ but typically more like $O(k * n)$, where $3 \leq k \leq 50$, in practical experience to date. Thus for every (S)CLP rulebase, there is an "indirect" (S)OLP version of it. However, the Courteous Compiler transform is sufficiently complex that it requires a considerable amount of cognitive effort (including time and mistakes) for a typical human rulebase author to achieve a similar result. Rather, it is typically much easier for the author simply to employ the Courteous Compiler by working in the courteous LP representation instead. That way, the rulebase author in effect lets the machine do the work (after all, that's what machines are for ...).

So far in this section we have been discussing how to use the Courteous feature of SCLP. In the rest of this section, we discuss how to use its *Situated* feature.

Example 4 (Ordering Lead Time Management, in SCLP)

Next, we extend Examples 1 and 2 to include actions and queries that are performed by procedural attachments — utilizing the *Situated* extension of (Courteous) LP. We add the following rules, effector statements, and sensor statements to our natural language specification:

- (Rule D:) "Accept a purchase request if it is received before the lead time."
- (Rule E:) "Deny a purchase request if it is received after the lead time."
- (Action Rule F:) "Update the orders database if a purchase request is accepted."
- (Action Rule G:) "Remind the customer of the lead time policy if a purchase request is denied."
- (Action Rule H:) "Notify the customer if a purchase request is accepted."
- (Action Rule I:) "Notify the customer if a purchase request is denied."

We represent these in the SCLP KR, in more detail, as follows:
$\langle d \rangle \quad acceptPurchaseRequest(?Request)$

negations of atoms.

$\leftarrow$ $orderModificationNotice(?Request, ?LeadTime) \wedge$
$deliveryDate(?Request, ?Day1) \wedge$
$receiptDate(?Request, ?Day2) \wedge$
$subtractDate(?Day1, ?LeadTime, ?Day3) \wedge$
$lessThanOrEqual(?Day2, ?Day3)$;

$\langle e \rangle$ $denyPurchaseRequest(?Request)$
$\leftarrow$ $orderModificationNotice(?Request, ?LeadTime) \wedge$
$deliveryDate(?Request, ?Day1) \wedge$
$receiptDate(?Request, ?Day2) \wedge$
$subtractDate(?Day1, ?LeadTime, ?Day3) \wedge$
$greaterThan(?Day2, ?Day3)$;

$\langle f \rangle$ $shouldUpdateOrderDb(?Request)$
$\leftarrow$ $acceptPurchaseRequest(?Request)$;

$\langle g \rangle$ $shouldRemindCustomerOfPolicy(?Request)$
$\leftarrow$ $denyPurchaseRequest(?Request)$;

$\langle h \rangle$ $shouldInformCustomer(?Request, accepted)$
$\leftarrow$ $acceptPurchaseRequest(?Request)$;

$\langle i \rangle$ $shouldInformCustomer(?Request, denied)$
$\leftarrow$ $denyPurchaseRequest(?Request)$;

The following *effector statements* each associate a pure-belief predicate, e.g., $shouldInformCustomer$, with an external procedure (here, a Java method), e.g., $orderMgmt.request.ack$. 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 $orderMgmt.request.ack$ is called with its parameters instantiated to $(request1049, accepted)$.

$shouldInformCustomer(?X, ?Y)$
:: e :: $orderMgmt.request.ack(?X, ?Y)$;

$shouldUpdateOrderDB(?Request)$
:: e :: $orderMgmt.request.updOrderDB(?Request)$;

$shouldRemindOfPolicy(?Request)$
:: e :: $orderMgmt.request.remind(?Request)$;

More SCLP Notation: here in an effector statement, from left to right, the predicate comes first as an open atom (in which that predicate appears), followed by ":: e ::", followed by the external procedure (in which the same logical variables appear as arguments).

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.

$receiptDate(?X, ?Y)$
$\quad :: s :: orderMgmt.request.getReceiptDate(?X, ?Y);$
$deliveryDate(?X, ?Y)$
$\quad :: s :: orderMgmt.order.getDelivDate(?X, ?Y);$
$subtractDate(?X, ?Y, ?Z)$
$\quad :: s :: utils.date.subtract(?X, ?Y, ?Z);$
$lessThanOrEqual(?X, ?Y)$
$\quad :: s :: utils.arith.lte(?X, ?Y);$
$greaterThan(?X, ?Y)$
$\quad :: s :: utils.arith.gt(?X, ?Y);$
$notEquals(?X, ?Y)$
$\quad :: s :: utils.arith.neq(?X, ?Y);$

More SCLP Notation: here in a sensor statement, from left to right, the predicate comes first as an open atom (in which that predicate appears), followed by "$:: s ::$", followed by the external procedure (in which the same logical variables appear as arguments).

The added rules above enable one to make use of information accessible through calls to (attached) sensor procedures, infer more conclusions, and generate actions via calls to (attached) effector procedures.

Terminology: We say that the collection of information accessible through calls to sensor procedures (i.e., "*sensing*") can be viewed as a "*virtual knowledge base*" of facts. We call these "virtual" facts or "sensor" facts. We say that the actions generated via calls to effector procedures (i.e., "*effecting*") are "*launched*" actions.

Next, we continue our example. Let us consider the particular ordering situations $pu34$ and $po56$. Suppose that the sensor call

$orderMgmt.request.getDeliveryDate(pu34, ?Day1);$

returns with binding 2003_06_23 for variable $?Day1$. This corresponds to the virtual fact

$deliveryDate(pu34, 2003\_06\_23);$

In like fashion, suppose the overall virtual knowledge base of (relevant) sensor facts consists of the following:

$deliveryDate(pu34, 2003\_06\_23);$
$receiptDate(pu34, 2003\_06\_11));$
$deliveryDate(pu56, 2003\_06\_30);$
$receiptDate(pu56, 2003\_06\_25));$
$subtractDate(2003\_06\_23, days14, 2003\_06\_09);$
$subtractDate(2003\_06\_30, days2, 2003\_06\_28);$
$greaterThan(2003\_06\_11, 2003\_06\_09);$

$lessThanOrEqual(2003\_06\_25, 2003\_06\_28)$;
$notEquals(days14, days2)$;
$notEquals(days14, days30)$;
$notEquals(days2, days30)$;

The predicate *notEquals* was used already in Example 2, and explicit facts were specified for it there. Here, it is specified as a sensor built-in, which corresponds to typical practice in commercial rule systems. The *notEquals* facts listed in Example 2 can thus be omitted from the explicit rulebase here.

Let us call $RB3$ the rulebase that includes the above rules and sensor statements and effector statements. Note $RB3$ includes $RB1$ from Example 1 except that the explicit notEquals facts from there are omitted here, since here they are virtual instead.

Then $RB3$ (together with the virtual knowledge base of sensor facts) entails the following additional conclusions about the ordering situations $pu34$ and $pu56$:

$RB3 \models denyPurchaseRequest(pu34)$;
$RB3 \models acceptPurchaseRequest(pu56)$;
$RB3 \models shouldInformCustomer(pu34, denied)$;
$RB3 \models shouldRemindCustomerOfPolicy(pu34)$;
$RB3 \models shouldInformCustomer(pu56, accepted)$;
$RB3 \models shouldUpdateOrderDb(pu56)$;

Each of the above entailed conclusions triggers effecting. $RB3$ thus launches the following actions (effector calls):

$RB3 \Rightarrow orderMgmt.request.ack(pu34, denied)$;
$RB3 \Rightarrow orderMgmt.request.remind(pu34))$;
$RB3 \Rightarrow orderMgmt.request.ack(pu56, accepted)$;
$RB3 \Rightarrow orderMgmt.request.updOrderDB(pu56))$;

Notation: Here, $\Rightarrow$ indicates launching of an action.

These additional entailed conclusions and launched actions are as desired. Request $pu34$, which has lead time 14 days, is denied since it arrives only 11 days ahead of its delivery date. The customer (EagerCo) is thus notified and reminded of the lead time policy. Request $pu56$, which has lead time 2 days, is accepted since it arrives 5 days ahead of its delivery date. The customer (BeaverCo) is thus notified and the (seller ElvesCo's) orders database is updated.

6 RuleML Syntax: DTD for SCLP

In this section, we specify the XML syntax for SCLP in RuleML. In so doing, we extend the syntax of RuleML syntax to SCLP – from the previous existing Horn Logic Programs (OLP) case ("sublanguage") of the syntax. This extension from Horn to SCLP is a novel contribution by us. This syntax specification takes the form of an XML Document Type Definition (DTD).

Below we give that current DTD for SCLP RuleML. (More precisely, this is version V0.8, dated April 22, 2003, posted at http://www.ruleml.org, "monolith" version without the query element.) [26] For the sake of brevity and clarity in presentation here, it has been altered to eliminate most comments and whitespace, and to rearrange the sequence of the statements, but not changed in substance.

Bold face indicates modifications made for the SCLP extension, i.e., differences from the Horn LP RuleML V0.8 DTD. We interleave some in-line comments, each enclosed in "/* ...*/" for clarity.

In RuleML's abstract syntax, only the elements that represent ordered tuples – i.e., "tup" and "bmtup" below – are intended to be sequence-sensitive. In order to maximize sequence-independence despite XML's sequence-sensitivity, in the DTD all permutations of the ordering of children are permitted for every other element. Unfortunately, the DTD mechanism does not permit one to state sequence-insensitivity in a more elegant manner than listing all the allowed permutations. However, the DTD mechanism has other virtues: it is simple, widely used, and widely familiar.

```
<!ELEMENT rulebase ( (_rbaselab, (imp | fact | mutex | sens | effe)*)
                          | ((imp|fact|mutex|sens |effe)+, _rbaselab?) )>
<!ELEMENT _rbaselab (ind | cterm)> /* rbaselab = rulebase label */
/* imp = implication rule ; rlab = rule label */
<!ELEMENT imp ( (_head, ((_body,_rlab?) | (_rlab,_body?))? )
                      | (_body, ((_head,_rlab?) | (_rlab,_head)))
                      | (_rlab,((_head,_body?) | (_body,_head))) )>
<!ELEMENT fact ( (_rlab,_head) | (_head,_rlab?) )>
<!ELEMENT _rlab (ind | cterm) >
<!ELEMENT _head (cslit | atom | andh)>
<!ELEMENT _body (fclit | atom | cslit | flit | andb | orb | and)>
<!ELEMENT andb ((fclit | atom | cslit | flit | andb | orb)*)>
<!ELEMENT and ((atom)*)>
```

[26] Our first version of this extension was in late 2001 and extended the then-current Horn RuleML version which was V0.7; that first version was then implemented in the first version of SweetRules.

<!ELEMENT orb ((**fclit** | atom | **cslit** | **flit** | andb | orb)+)>
<!ELEMENT andh ((**cslit** | atom | andh)+)>
<!ELEMENT atom ((_opr, (ind | var | cterm)*) | ((ind|var|cterm)+, _opr))>
<!ENTITY % bool "yes|no">
<!ELEMENT flit ((_opr, (ind|var|cterm)*) | ((ind|var|cterm)+, _opr))>
/* fneg = negation-as-failure (NAF) sign ; lit = literal */
<!ATTLIST flit fneg (%bool;) #IMPLIED>
<!ELEMENT _opr (rel)>
<!ELEMENT rel (#PCDATA)> /* rel = relation = predicate */
<!ATTLIST rel href %URI; #IMPLIED>
<!ENTITY % URI "CDATA">
<!ELEMENT var (#PCDATA)> /* var = (logical) variable */
<!ELEMENT ind (#PCDATA)> /* ind = individual */
<!ATTLIST ind href %URI; #IMPLIED>
<!ELEMENT cterm ((_opc, (ind | var | cterm | tup | roli)*)
| ((ind | var | cterm | tup | roli)+, _opc))>
<!ELEMENT _opc (ctor)>
<!ELEMENT ctor (#PCDATA)>/* ctor = constructor = logical function*/
<!ATTLIST ctor href %URI; #IMPLIED>
<!ELEMENT tup ((ind | var | cterm | tup | roli)*)>
<!ELEMENT roli ((_arv)*)> /*tup = tuple (ordered); roli = role'd list*/
<!ELEMENT _arv ((arole, (ind | var | cterm | tup | roli))
| ((ind | var | cterm | tup | roli), arole)) >
<!ELEMENT arole (#PCDATA)> /* arole = argument role = slot */
<!ATTLIST arole href %URI; #IMPLIED>
<!ATTLIST rulebase direction/* = advisory intended inferencing direction*/
(forward | backward | bidirectional) "bidirectional">
/* Next two attributes optionally specify a referenced XML Schema */
<!ATTLIST rulebase xsi:noNamespaceSchemaLocation %URI; #IMPLIED>
<!ATTLIST rulebase xmlns:xsi %URI; #IMPLIED>
/* Syntax specifically for COURTEOUS follows */
<!ELEMENT fclit ((_opr, (ind|var|cterm)*) | ((ind|var|cterm)+, _opr))>
<!ATTLIST fclit cneg (%bool;) #IMPLIED>
<!ATTLIST fclit fneg (%bool;) #IMPLIED>
/* cneg = (limited) classical/strong negation sign */
<!ELEMENT cslit ((_opr, (ind|var|cterm)*) | ((ind|var|cterm)+, _opr))>
<!ATTLIST cslit cneg (%bool;) #IMPLIED>
<!ELEMENT mutex ((_oppo, _mgiv?) | (_mgiv, _oppo))>
<!ELEMENT _oppo (ando)>
<!ELEMENT _mgiv (fclit | atom | flit | cslit | andb | and | orb)>
/* mgiv = mutex's "given" part/condition */
<!ELEMENT ando (cslit, cslit)>
/* Syntax specifically for SITUATED follows */
<!ELEMENT sens ((_opr, ((_aproc, _modli?) | (_modli,_aproc)))
| (_aproc, ((_opr,_modli?) | (_modli,_opr)))

```
                    | (_modli,((_aproc,_opr) | (_opr,_aproc))) )>
/* sens = sensor statement ; effe = effector statement */
/* sensor procedure declaration has a required binding pattern:
  modli = mode list ; mode = bound vs. free -ness of argument*/
<!ELEMENT effe ((_opr, _aproc) | (_aproc, _opr))>
<!ELEMENT _aproc (jproc | uproc)>
/* aproc = attached procedure ; j = Java */
<!ELEMENT jproc ((clas, ((meth, path?) | (path, meth)))
                    | (meth, ((clas, path?) | (path, clas)))
                    | (path, ((meth, clas) | (clas, meth))))>
<!ELEMENT uproc (#PCDATA)>
<!ATTLIST uproc href %URI; #IMPLIED>
/* u = "universal", i.e., general-purpose for using Web protocol,
      e.g., Web Services – to be refined in future */
<!ELEMENT path (#PCDATA)>
<!ATTLIST path href %URI; #IMPLIED>
<!ELEMENT clas (#PCDATA)> /* clas = class */
<!ATTLIST clas href %URI; #IMPLIED>
<!ELEMENT meth (#PCDATA)> /* meth = method */
<!ATTLIST meth href %URI; #IMPLIED>
<!ELEMENT bmode EMPTY> /* bmode = (binding) mode */
<!ATTLIST bmode val (%bind;) "free">
<!ENTITY % bind "bound|free">
<!ELEMENT _modli ((bmode | bmtup | bmroli)*)>
<!ELEMENT bmtup ((bmode | bmtup | bmroli)*)>
<!ELEMENT bmroli ((_arbm)*)>
<!ELEMENT _arbm ((arole, (bmode | bmtup | bmroli))
                    | ((bmode | bmtup | bmroli), arole) )>
```

Observe that the DTD is nicely short overall: about two pages (86 lines including comments) in this paper's format. This relatively manageable size provides a degree of cognitive simplicity that thereby aids implementers, (rule) authors, and users.

We also have developed an abstract syntax (specification) for SCLP RuleML that corresponds to the XML DTD (concrete) syntax. More details about these, including extensive design comments, are available at http://www.ruleml.org, http://ebusiness.mit.edu/bgrosof/#RuleML, and in the email archives of the Joint US/EU ad hoc Agent Markup Language Committee (a.k.a. the "Joint Committee") at http://www.daml.org/committee.

In current work, we are extending the existing DTD version of the XML syntax specification for RuleML, including for SCLP RuleML, to other forms of concrete syntax. This work is in collaboration with others in the RuleML Initiative (especially Harold Boley, Said Tabet, and Michael Dean). Our abstract

syntax supports these other forms. One form employs XML Schema, a generalization of DTD, to define the syntax. Another form encodes the syntax using Resource Description Format (RDF). RDF is a form of semi-structured data that is somewhat more shareable than XML and that has a somewhat semantically cleaner data model than XML has, but is typically itself in turn encoded syntactically using XML ("RDF-XML"). A third form of concrete syntax is a concise one in non-XML ASCII, intended for human reading and editing. This new ASCII syntax is based in part upon the existing ASCII "SCLPfile" syntaxes of SweetRules and IBM CommonRules.

Example 5 (Example of a SCLP Rulebase in RuleML Syntax)

Next we give a basic example of a SCLP rulebase in RuleML, i.e., in its current (V0.8) XML syntax. RuleML inherits the relatively verbose character of XML. For the sake of brevity, we show in detail only the first rule — Rule A from Example 2.

```
<?xml version="1.0" ...>
<!DOCTYPE rulebase SYSTEM ...>
<rulebase>
<imp>
  <_rlab><ind>a</ind></_rlab>
  <_head>
    <atom>
      <_opr><rel>orderModificationNotice</rel></_opr>
      <var>Request</var>
      <ind>days14</ind>
    </atom>
  </_head>
  <_body>
    <andb>
      <atom>
        <_opr><rel>preferredCustomerOf</rel></_opr>
        <var>Buyer</var>
        <var>Seller</var>
      </atom>
      <atom>
        <_opr><rel>purchaseRequest</rel></_opr>
        <var>Request</var>
        <var>Buyer</var>
        <var>Seller</var>
      </atom>
    </andb>
  </_body>
</imp>
```

```
...
</rulebase>
```

More examples of RuleML rulebases are available via the author's website and the RuleML website.

7 SweetRules 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 SweetRules. "SWEET" is short for "Semantic WEb Enabling Technology". As we mentioned earlier, the first version of SweetRules was a refereed system demonstration at the Workshop on Information Technology and Systems (WITS '01) in Dec. 2001. The version of SweetRules demonstrated there implemented fully general Courteous LP, but not yet most Situated features. The current version of SweetRules supports Situated features to a considerable degree, e.g., as part of its SweetJess component [8]. Next, we give an overview of that prototype and its architecture.

The SweetRules toolset consists of three kinds of components: translators, inferencing engines, and front-ends.

Translators translate between different rule languages / systems, using RuleML as an interchange format. Currently there are already bidirectional translators from SCLP RuleML to the following rule languages / systems:

- IBM CommonRules system, via its BRML XML syntax for SCLP rules. This is a semantically clean forward-inferencing SCLP inferencing system, that also has translation capabilities (more about that below). The CommonRules project team participates in the RuleML Initiative and has announced they plan to support RuleML in future.
- XSB Prolog logic programming system [27] . This is a backward-direction Ordinary LP inferencing system. It is free, open source, Web-downloadable [28] , and has an (ODBC) backend to SQL database systems such as Oracle's etc. Implemented in C, and using tabling, XSB is fast and scales well to very large rulebases + databases. It is semantically clean and expressively powerful. [29]

[27] by David Warren and his group at State University of New York at Stonybrook: http://www.sunysb.edu/~sbprolog
[28] http://xsb.sourceforge.net
[29] w.r.t. recursion in that it supports a quite broad case of the well founded semantics [15] for negation-as-failure in Ordinary LP (and thus in Courteous LP).

- Jess ("Java Expert Systems Shell") production rules system, which has an ASCII syntax. Jess is a popular, semi-open-source OPS5-style inferencing system, implemented in Java.[30] The component of SweetRules that translates to Jess and does inferencing via Jess is called SweetJess; see [8] and http://daml.umbc.edu/sweetjess for details.
- Knowledge Interchange Format (KIF) language. This is an ASCII syntax for first-order logic (FOL), and thus can represent Horn rules. There are several rule systems that perform inferencing in KIF, e.g., JTP.[31] As we discussed earlier, the close successor of KIF is (Simple) Common Logic (SCL). SCL is currently developing an XML syntax, largely influenced by RuleML's.
- Smodels[32]. This is an open-source forward-direction OLP inferencing system, with a Prolog-style syntax, implemented in C.
- SCLPfile format, a concise Prolog-like ASCII syntax for SCLP. This is useful for manual editing of rulebases (a more limited version of this is supported directly in IBM CommonRules).
- Situated *Ordinary* LP RuleML – via a Courteous Compiler. As we mentioned earlier, the Courteous Compiler "compiles away" the courteous expressive features from a SCLP and generates a semantically equivalent (Situated) Ordinary LP.

Inferencing engines perform inferencing in RuleML. Inferencing starts from a set of premises expressed in RuleML, i.e., a rulebase, and generates conclusions also expressed in RuleML. The toolset has capabilities both for forward-direction inferencing and for backward-direction answering, i.e., query-answering. SweetRules performs RuleML inferencing indirectly, in three steps:

(1) translating the given rulebase to the rule language of another rule system S. S might be Jess or CommonRules, for example. Likewise, it translates the given query, if doing backward-direction reasoning, to S.
(2) employing S to do inferencing on the resulting rulebase.
(3) translating back to RuleML the results of inferencing in S.

Front-ends provide API's and user interface (UI) capabilities to

- issue commands: to do translation or inferencing, including by composing various sub-components of SweetRules along with other available rule systems and tools.
- edit: rulebases, the results of translation, or the results of inferencing.

[30] by Ernest Friedman-Hill of Sandia National Labs, USA: http://herzberg.ca.sandia.gov/jess
[31] by Gleb Frank, Richard Fikes and Jessica Jenkins, at Stanford University: http://www.ksl.stanford.edu/software/JTP
[32] by Ilkka Niemela and Patrik Simons of University of Helsinki, Finland: http://saturn.hut.fi/html/staff/ilkka.html

In current work, further additions to SweetRules's functionality are being developed by our research group, including to integrate the tools more smoothly, to support translation to additional rule systems, and to support new expressive features and forms of syntax as those evolve from the RuleML Initiative and the Semantic Web more generally.

SweetRules is implemented in Java and XSLT [33], and uses several other tools, including as discussed above. In particular, the translator from RuleML to IBM CommonRules' BRML is implemented in XSLT. SweetRules makes use of several components from CommonRules, including its Courteous Compiler and its translators to XSB, KIF, and Smodels. IBM CommonRules (its version of 1999-2000; it has more recent versions since) was the forerunner to SweetRules. No accident – we conceived CommonRules and led its design, implementation, and application piloting while then at IBM Research.

To recap: overall, SweetRules supports translation among and inferencing in two of the four major currently commercially important (CCI) families of rule systems that we discussed earlier, plus in FOL via KIF. It does so either by direct translation, as in the case of SweetJess, or by indirect translation through CommonRules. A free, open-source, Web-downloadable version of SweetRules is planned for the near future.

8 Conclusions, Current and Future Work

The Abstract provided an initial overview of our contributions reported here. In previous work, we developed the SCLP KR. Here, we showed for the first time how to extend RuleML to support SCLP, by providing (1) a DTD specification of its syntax, and (2) a prototype toolset — SweetRules – for translation and inferencing with SCLP RuleML. We gave a novel detailed long example of how to use SCLP RuleML to represent e-commerce rules. The example rules were for ordering lead time management, which is a key task in many B2B settings including manufacturing supply chain management (SCM). We gave a novel discussion of the features and advantages of overall RuleML and of SCLP RuleML in particular, focusing especially on the Courteous and Webizing aspects, including in the context of the long example.

Our previous papers about the SCLP KR include a number of other examples of using it to represent e-commerce rules, especially for e-contracting (including negotiation) [11,14,12,8]. For example, rules are useful to represent product/service/deal descriptions; pricing; search, selection, monitoring, and

[33] eXtended Stylesheet Language Transformations, a popular tool for translating between different XML representations; see, e.g., http://www.w3.org/TR/xslt

composition of Web Services. We have previously in [11] given a detailed requirements analysis which motivates and justifies use of Webized SCLP, and thus SCLP RuleML, to represent e-commerce rules for e-contracting. In current work, we are exploring use of SCLP RuleML to represent e-contracting aspects of Semantic Web Services, including via our participation in the Semantic Web Services Initiative [34] . Interesting relevant Web Services standards efforts include UDDI for discovery and BPEL4WS for business process interactions, both from Oasis [35] . Related also is our previous work on using multi-agent LP rules to represent trust management policies and reasoning, including for security authorization involving delegation [13].

Earlier we discussed several directions of current and future work, notably on the SweetRules prototype and on development of more forms of syntax for SCLP RuleML, including to connect it to Web Services more tightly. Next we discuss additional directions for current and future work.

One direction for future work is thus to explore more pilot applications of SCLP RuleML in e-commerce. An interesting area is to employ SCLP RuleML within emerging general-purpose Web e-commerce communication approaches such as ebXML [36] and its follow-on UBL [37] . We hope a community of researchers and practitioners will try out the SCLP RuleML approach in various areas of e-commerce, develop requirements, and provide a feedback loop to further improvements and extensions.

We have already identified, however, several important directions for future work on the foundations of SweetRules approach, to extend the kinds of rule languages / systems to (and from) which SCLP RuleML can be translated and in which SCLP RuleML inferencing can be performed. One direction is the CCI family of SQL databases. Another direction is the CCI family of Event-Condition-Action rules. A third direction is to support OWL ontologies, i.e., the Description Logic KR which is a fragment of FOL. This is needed in order to semantically integrate ontologies so that RuleML rules can make use of predicates defined in OWL ontologies while ensuring completeness and consistency w.r.t. the semantics of those ontologies [12,9]. A fourth direction for future work is to support the emerging standards for semi-structured databases encoded in XML: XQuery, the new W3C standard for XML databases, and the as-yet loose area of RDF query languages / systems. In current work, we with collaborators are pursuing aspects of all these four directions. Each of these four directions requires some degree of new theoretical research. In particular, we (with student Boris Motik of University of Karslruhe, Germany) have developed a running prototype system, called SweetOnto, that translates from

[34] http://www.swsi.org
[35] http://www.oasis-open.org
[36] http://www.ebxml.org
[37] www.oasis-open.org/committees/ubl

a subset of OWL to Horn RuleML by implementing the Description Logic Programs approach [9], and plan to provide a free public Web-downloadable version of it in the near future.

Acknowledgements

Harold Boley and Said Tabet are my close collaborators in leading the RuleML Initiative. Hoi Chan of IBM T.J. Watson Research Center contributed much to the the implementation of IBM CommonRules including design details of Business Rules Markup Language. Mohammed Youssef Kabbaj, student at MIT, contributed to the initial SweetRules prototype's XSLT implementation. Earl J. Wagner, student at MIT, contributed to the front-end components of SweetRules. Funding support was provided by grants from the DARPA Agent Markup Language (DAML) program and the Center for eBusiness @ MIT Vision Fund.

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. Grosof. 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. Grosof. 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. Grosof. 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. Grosof. 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. Grosof. 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 ; 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 a MIT report; see author's webpage. 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. Grosof. Representing E-Business Rules for the Semantic Web: Situated Courteous Logic Programs in RuleML. In Jeffrey Parsons and Olivia Sheng, editors, *Proceedings of the 11th Workshop on Information Technologies and Systems (WITS '01)*, 2001. (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). Available at author's website, along with extended report version of Dec. 2002.

[8] Benjamin N. Grosof, Mahesh D. Gandhe, and Timothy W. Finin. SweetJess: Translating DamlRuleML to Jess. In *Proc. International Workshop on Rule Markup Languages for Business Rules on the Semantic Web*, 2002. (http://tmitwww.tm.tue.nl/staff/gwagner/RuleML-BR-SW.html) Held 14 June 2002, Sardinia (Italy) in conjunction with the First International Semantic Web Conference (ISWC-2002). Extended and updated Working Paper available at first author's website. Prototype available via http://www.daml.umbc.edu/sweetjess.

[9] Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description Logic Programs: Combining Logic Programs with Description Logic. In *Proceedings of the 12th International Conference on the World Wide Web (WWW-2003)*. ACM Press, 2003. (http://www.www2003.org) Held May 20–23, 2003, Budapest, Hungary.

[10] Benjamin N. Grosof and Yannis Labrou. An Approach to using XML and a Rule-based Content Language with an Agent Communication Language. In Frank Dignum and Mark Greaves, editors, *Issues in Agent Communication*. Springer-Verlag, 2000. Slightly revised from version in: Proc. IJCAI-99 Workshop on Agent Communication Languages (ACL-99) (http://wwwis.win.tue.nl/acl99), available also as IBM Research Report RC 21491 (May 1999).

[11] Benjamin N. Grosof, 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.

[12] Benjamin N. Grosof and Terrence C. Poon. SweetDeal: Representing Agent Contracts with Exceptions using XML Rules, Ontologies, and Process Descriptions. In *Proceedings of the 12th International Conference on the World Wide Web (WWW-2003)*. ACM Press, 2003. (http://www.www2003.org) Held May 20–23, 2003, Budapest, Hungary. Extended version submitted to journal is available at author's website.

[13] Ninghui Li, Benjamin N. Grosof, and Joan Feigenbaum. Delegation logic: A logic-based approach to distributed authorization. *ACM Transactions on Information Systems Security (TISSEC)*, 6(1), Feb. 2003.

[14] Daniel M. Reeves, Michael P. Wellman, and Benjamin N. Grosof. Automated negotiation from declarative contract descriptions. *Computational Intelligence*, 18(4):482–500, Nov. 2002. Special issue on Agent Technologies for Electronic Commerce.

[15] A. Van Gelder, K. Ross, and J. Schlipf. The well-founded semantics for general logic programs. *Journal of ACM*, 38(3):620–650, 1991.