A Declarative Approach to Business Rules in Contracts: Courteous Logic Programs in XML *

Benjamin N. Grosof
IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598, USA
grosof@us.ibm.com (alt.: grosof@cs.stanford.edu)
http://www.research.ibm.com/people/g/grosof/

Yannis Labrou
Computer Science and Electrical Engineering and Department
University of Maryland, Baltimore County, Baltimore, MD 21250, USA
jklabrou@cs.umbc.edu
http://www.cs.umbc.edu/~jklabrou/

Hoi Y. Chan
IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598, USA
hychan@us.ibm.com

Abstract

We address why, and especially how, to represent business rules in e-commerce contracts. By contracts, we mean descriptions of goods and services offered or sought, including ancillary agreements detailing terms of a deal. We observe that rules are useful in contracts to represent conditional relationships, e.g., in terms & conditions, service provisions, and surrounding business processes, and we illustrate this point with several examples.

We analyze requirements (desiderata) for representing such rules in contracts. The requirements include: declarative semantics so as to enable shared understanding and interoperability; prioritized conflict handling so as to enable modular updating/revision; ease of parsing; integration into WWW-world software engineering; direct executability; and computational tractability.

We give a representational approach that consists of two novel aspects. First, we give a new fundamental knowledge representation formalism: a *generalized version of Courteous Logic Programs* (CLP), which expressively extends declarative ordinary logic programs (OLP) to include prioritized conflict handling, thus enabling modularity in specifying and revising rule-sets. Our approach to implementing CLP is a *courteous compiler* that transforms any CLP into a semantically equivalent OLP with moderate, tractable computational overhead.

Second, we give a new *XML encoding* of CLP, called Business Rules Markup Language (BRML), suitable for interchange between heterogeneous commercial rule languages. BRML can also express a broad subset of ANSI-draft Knowledge Interchange Format (KIF) which overlaps with CLP.

Our new approach, unlike previous approaches, provides not only declarative semantics but also prioritized conflict handling, ease of parsing, and integration into WWW-world software engineering. We argue that this new approach meets the overall requirements to a greater extent than any of the previous approaches, including than KIF, the leading previous declarative approach.

We have implemented both aspects of our approach; a free *alpha prototype* called *CommonRules* was released on the Web in July of 1999, at http://alphaworks.ibm.com.

An extended version of this paper will be available as a forthcoming IBM Research Report (at http://www.research.ibm.com).

1 Introduction

One form of commerce that could benefit substantially from automation is contracting, where agents form binding, agreeable terms, and then execute these terms. By "agents", we mean initially parties in the Economics sense, e.g., as buyers or sellers; however, once automated, these agents might be intelligent agents in the Computer Science sense.

By e-commerce “contracts”, we mean in a broad sense: descriptions of goods and services offered or sought, along with applicable terms & conditions, i.e., ancillary agreements detailing terms of a deal. Such terms include customer service agreements, delivery schedules, conditions for returns, usage restrictions, and other issues relevant to the good or service provided.

Such descriptions are to be found in catalogs and storefronts, as well as in bids and requests communicated (e.g., by agents) during negotiations, procurements, and auctions.

The overall contracting process comprises several stages, including broadly:

1. *Discovery*: Agents find potential contracting partners.
2. *Negotiation*: Contract terms are determined through a process of communication between agents, often involving iterative modification of the contract terms.
3. *Execution*: Transactions and other contract provisions are executed by the agents.

We observe that many contract terms involve conditional relationships, and can conveniently be expressed as rules, often called *business rules* or business policies. (Sometimes, “business rule” is used to mean any kind of significant application logic, e.g., the algebraic formula for computing an insurance annuity. By “rule” in this paper, we mean more specifically an implication (i.e., IF-THEN) in which the antecedent (i.e., the IF part) may contain multiple conjoined (i.e., AND’ed) conditions.)

For example, rules are useful in describing:

- terms & conditions, e.g., rules for price discounting;
- service provisions, e.g., rules for refunds; and
- surrounding business processes, e.g., rules for lead time to place an order.

(Section 2 elaborates on these examples.) We believe that rules as an overall representational approach capture well many aspects of what one would like to describe in automated contracts.

In this work we are concerned primarily with the negotiation stage of contracting, and specifically with how to represent business rules in contracts.

The contract terms may or may not be modified during the negotiation stage. If not, the negotiation stage may be a relatively simple process of communication followed by acceptance. Crucial during negotiation, however, is that all agents must understand and agree. In terms of automation, the contract has to be communicated and digested by each agent, with shared semantics. (More generally, some agents might be human, as well as automatic; however, the aspects of human interaction are beyond the scope of this paper.)

Our goal is a shared language with which agents can reach a common understanding of rules in contracts, such that the rules are relatively easily modifiable, communicatable, and executable by the agents.

Note that we make a sharp distinction between the representational mechanism for communicating contract rules, and the actual rule execution mechanisms employed by participating agents. Our concern here is with the former, though of course in designing a communication mechanism one must consider and anticipate the requirements of execution to be performed by each of the agents.

We will take a *declarative* approach to business rules in contracts. By “declarative” semantics for rules, we mean in the sense of knowledge representation (KR) theory, in a manner abstracted away from the choice of implementation approach: the semantics say which conclusions are entailed (e.g., model-theoretically) by a given set of premises, without dependence on procedural or control aspects of inference algorithms. In particular, declarative semantics abstract away from whether the direction of rule inferencing is backward versus forward.

2 Example Roles for Rules and Prioritized Conflict Handling

Next, we give a few brief examples, expressed in natural language, of business rules in contracts. As we go, we will see both the need for prioritized conflict handling — because of conflict, and the opportunity for prioritized conflict handling — because of naturally available prioritization information.

Example 1 (Refund Policy)
A typical example of a seller’s refund policy is:

- (Rule A:) “If buyer returns the purchased good for any reason, within 30 days, then the purchase amount, minus a 10% restocking fee, will be refunded.”
- (Rule B:) “If buyer returns the purchased good because it is defective, within 1 year, then the full purchase amount will be refunded.”
- (Priority Rule:) “If both Rules A and B apply, then Rule B ‘wins’, i.e., the full purchase amount will be refunded”.

Here, we say a rule “applies” when that rule’s body (i.e., antecedent) is satisfied. The Priority Rule is necessary because it can happen that the buyer returns the good because it is defective within 30 days. In this case, Rules A and B *conflict* with each other; both rules apply but their consequents are incompatible with each other. The conflict is resolved by *priority*: Rule B is specified to have higher priority than Rule A.

Example 2 (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 3 (Bookstore Personalized Discounting)
Sellers often provide personalized price discounting. A topic of considerable interest today among e-commerce sellers is how to do this more cheaply and uniformly by automating it. An example of a bookstore's personalized discounting policy is:

- (Rule A:) "5 percent discount if buyer has a history of loyal spending at the store."
- (Rule B:) "10 percent discount if buyer has a history of big spending at the store."
- (Rule C:) "5 percent discount if buyer has a store charge card."
- (Rule D:) "10 percent discount if buyer is a member of the store's Platinum Club."
- (Rule E:) "No discount if buyer has a late-payment history during the last year."
- (Priority Rules:) "B is higher priority than all the others; D is higher priority than A and than C; E is higher priority than A and than C and than D."

Here, the Priority Rules specify a set of priority comparisons, that are sufficient to resolve all the potential conflicts among the rules.

More Example Roles for Rules

We believe that many contract terms can be well represented as rules.

- Policies about **cancelling orders** are often similar in feel to the examples above about refunds and lead time.
- Policies about **discounting for groups** or preferred customer organizations, e.g., hotel discounts for AAA members, are often similar in feel to the example above about personalized discounting.
- In supply chain settings: requests for proposals, and responding bids, often involve **conditional relationships between price, quantity, and delivery date**, e.g., "If the order quantity is between 400 and 1000 units, and the delivery date is between 15 and 30 days from the order date, then the price is $48 per unit.".
- In **product catalogs**, many properties of a product are most naturally specified as conditional on other properties of that product, rather than being particular to an individual product. E.g., "all laptop computers include an internal modem". E.g., "all women's T-shirts are available in sizes XS, S, M, and L", "all men's T-shirts are available in sizes S, M, L, XL, and XXL", and "all T-shirts are available in colors navy, khaki, and black". E.g., "all V-neck sweaters are available in fabrics acrylic and cashmere".
- Policies about **creditworthiness, trustworthiness, and authorization** are often naturally expressed in terms of sufficient and/or necessary conditions. Such conditions include: credentials, e.g., credit ratings or professional certifications; third-party recommendations; properties of a transaction in question, e.g., its size or mode of payment; and historical experience between the agents, e.g., familiarity or satisfaction.

3 Rule Representations: Previous Approaches including KIF, Requirements Analysis

In section 1, we stated that in this work our goal is a shared language with which agents can reach a **common understanding** of rules in contracts, such that the rules are relatively easily **modifiable**, **communicatable**, and **executable** by the agents.

In this section, we analyze and elaborate these requirements (i.e., criteria or desiderata, which we bold-face throughout this section) for a rule representation, and discuss candidate previous approaches in light of them, as motivation for our approach. We begin by reviewing previous approaches.

There are multiple approaches already in wide implemented commercial use for representing business rules generally (not for contracts in particular). One approach is directly as *if-then code constructs* in general-purpose imperative programming languages such as C, C++, and Java[1]. Other approaches are more specific to rules. A second approach is *Prolog* [3], a programming language oriented towards backward-chaining rules. A third approach, closely related to Prolog, is *SQL views*. In SQL-type relational database

[1]trademark of Sun Microsystems Inc.

systems [22] a view is defined, in effect, by a set of rules. A fourth approach, fairly closely related to SQL views, is *event-condition-action rules* / "active rules" / "triggers" [22] of the kind found in many database systems and routing systems. These are forward-directed, triggered by events such as updating a database relation. A fifth approach is *production rules*, a form of forward-chaining rules, of the kind in systems descended from the OPS5 system [4]. There are other approaches as well, e.g., in expert systems, knowledge-based systems, and intelligent agent building tools, but the above-listed are probably the currently most commercially important, especially in e-commerce.

A sixth approach for representing business rules is *Knowledge Interchange Format (KIF)*[2]. KIF is not a directly executable representation. Rather, KIF is a language for interchange of knowledge, including rules, between heterogeneous software systems, e.g., agents. More precisely, KIF is a prefix[3] version of first-order predicate calculus (i.e., first-order classical logic) with extensions to support the "quote" operator (thus enabling additional expressiveness akin to that of classical higher-order logic) and definitions. KIF is currently well along in the ANSI standards committee process. Supporting or endorsing KIF is also being considered informally in several other standards efforts relevant to agent communication, e.g., FIPA[4]. KIF, however, is not yet in wide implemented commercial use.

Next, we elaborate on requirements and relate them to the previous approaches. One group of requirements revolves around heterogeneity. Specifically, the multiplicity of widely implemented approaches implies an immediate requirement to **cope with heterogeneity of implementations** of business rules. The ability to communicate with shared understanding then implies the requirements of **interoperability** with **declarative semantics**. The above-listed widely implemented approaches lack fully declarative semantics.

Communicatability and interoperability imply the requirement of **ease of parsing** rule-sets that are being communicated. Interoperability and practical executability imply the requirement of **integration into WWW-world software engineering** overall. The XML aspect of our approach facilitates such parsing and integration.

A second group of requirements revolves around expressiveness. A basic overall requirement is **expressive power** in specifying rules. Practical executability, however, implies the strong desire for **computational tractability** in the sense of average-case and worst-case complexity. Expressive power has to be balanced against tractability.

[2]`http://logic.stanford.edu/kif/` and `http://www.cs.umbc.edu/kif/`

[3]The current draft ANSI specification of KIF (`http://logic.stanford.edu/kif/dpans.html`) also includes an infix version of KIF intended for human consumption rather than automated exchange.

[4]Foundation for Intelligent Physical Agents: `http://www.fipa.org`

Ease of modifiability implies the requirement of **expressive convenience** for the process of specification. Expressive convenience is in part an aspect of expressive power, but also implies the need for **conceptual naturalness of semantics**.

An important aspect of expressive power is the ability to conveniently express rules that are **logically non-monotonic**, i.e., rules that employ *negation-as-failure* or are *default rules*. In particular, it is important to conveniently express **prioritized conflict handling**, i.e., where some rules are subject to *override by higher-priority conflicting rules*, such as by special-case exceptions, by more-recent updates, or by higher-authority sources. As we saw in our contract rule-set examples in section 2, conflict, and the need/opportunity for prioritized override, frequently arise.

Most commercially important rule systems (including the above-listed widely implemented approaches) employ non-monotonic reasoning as an essential, highly-used feature. Typically, they employ some form of negation-as-failure. Often they employ some form of prioritized override between rules, e.g., the static rule sequence in Prolog or the computed rule-activation sequence/"agenda" in OPS5-heritage production rule systems.

In modern software design, it is widely recognized that a vital aspect of modifiability is **modularity and locality in revision**, e.g., in the manner that subclassing and information hiding provide in object-oriented programming. Prioritized conflict handling enables significantly easier modification and more modular revision/updating. New behavior can be specified more often by simply adding rules, without needing to modify the previous rules. E.g., a more specific-case rule can be added and given higher-priority than a previous general-case rule, without needing to modify the general-case rule. This is similar to the modular/local gracefulness in object-oriented programming of adding a subclass without needing to modify the superclass.

Another important aspect of expressive power, in relation to practical executability, is the ability to conveniently express **procedural attachments**. Procedural attachments are the association of procedure calls with belief expressions, e.g., in Example 1, the association of the Java method `CustomerAccount.setCreditAmount` with the logical predicate $refund$. Procedural attachments are crucial in order for rules to have actual effect beyond pure-belief inferencing, e.g., for actions to be invoked/performed as a result after rule conclusions are inferred. Procedural attachments are also very useful to invoke procedure calls when rule conditions are tested/queried. Almost all of the above-listed widely implemented approaches include procedural attachments.

However, procedural attachments are semantically problematic, and an open research issue today is how to give them a semantics that abstracts away from implementation details and is thus akin to being declarative. (See the discussion of Situated Logic Programs in section 5.)

KIF has been developed specifically for purposes of communication, in response to the need to cope with implementational heterogeneity of rules and other knowledge. By contrast with the above-listed widely implemented approaches, KIF has a fully declarative semantics; indeed, that is its main intended strength. Its declarative semantics is based on classical logic, primarily focusing on first-order logic. First-order logic is logically monotonic, highly expressive, and computationally intractable (to perform inferencing) for the general case.

KIF can express a broad class of rules. However, it has several important shortcomings as a language for business rules in e-commerce, including in contracts. Perhaps most crucially, KIF has a shortcoming of its fundamental knowledge representation (KR): it cannot (conveniently) express logical non-monotonicity, including negation-as-failure, default rules, and prioritized conflict handling; KIF is logically monotonic. KIF also cannot (conveniently) express procedural attachments; it is a pure-belief KR. KIF has been designed with an orientation towards knowledge as a non-executable specification as much or more than towards knowledge as executable. Also, the KIF effort has focused more on a highly inclusively expressive representation than on ease of developing translators in and out of that representation (this is something the XML aspect of our approach improves upon).

In our view, none of the above-listed previous approaches to representing business rules, nor any other previous approach that we are aware of, satisfactorily meets the whole set of requirements we listed above (in bold-face). In particular, the widely-implemented approaches above lack sufficiently declarative semantics; KIF, though declarative, lacks logical non-monotonicity.

KIF's declarative approach does, however, inspire us to develop our own declarative approach.

4 A New Declarative Approach to Rules: Courteous Logic Programs + XML

The approach is to choose Ordinary Logic Programs (OLP) — in the declarative sense ([1] provides a helpful review), not Prolog — as a fundamental KR, plus to embody this KR concretely in XML for purposes of communication between the contracting agents. [5] Logic programs are not only relatively powerful expressively, but also practical, relatively computationally efficient, and widely deployed.

We expressively extend the approach's declarative fundamental KR formalism to be *Courteous Logic Programs (CLP)*. CLP expressively extends Ordinary Logic Programs to include prioritized conflict handling, while maintaining computational tractability. CLP is a previous KR [10] which we have here further expressively generalized, notably to handle *pairwise mutual exclusion conflicts*, rather than simply conflicts of the form p versus $\neg p$.

4.1 First KR Step: Ordinary Logic Programs

We begin defining our approach by choosing the fundamental KR for rules to be: declarative Ordinary Logic Programs (OLP), with the well-founded semantics (WFS) [23], initially expressively restricted to the *predicate-acyclic* case (defined below). (Henceforth, we will leave "declarative" implicit.) This is a "pure-belief" KR, i.e., it lacks procedural attachments.

Each rule in an OLP has the form:

$$A_0 \;\leftarrow\; A_1 \wedge \ldots \wedge A_m \wedge {\sim}A_{m+1} \wedge \ldots \wedge {\sim}A_n.$$

Here, $n \geq m \geq 0$, and each A_i is a logical atom. A_0 is called the *head* (i.e., consequent) of the rule; the rest is called the *body* (i.e., antecedent) of the rule. If the body is empty, the $\leftarrow$ may be omitted. A ground rule with empty body is called a *fact*. $\sim$ stands for the *negation-as-failure* operator symbol, and is read in English as "fail". Intuitively, ${\sim}p$ means that p is not believed to be $true$, i.e., that p's truth value is *either* $false$ *or* $unknown$.

E.g., the first rule in Example 2 might be written as:

$$orderModificationNotice(?Order, days14) \\ \leftarrow\; preferredCustomerOf(?Buyer, ?Seller) \\ \wedge\, purchaseOrder(?Order, ?Buyer, ?Seller).$$

Here, the prefix "?" indicates a logical variable.

Ordinary LP's have been well-studied, and have a large literature (reviewed, for example, in [1]). For several broad but restricted expressive cases, their (declarative) semantics is uncontroversial: e.g., for the predicate-acyclic, stratified, locally stratified, and weakly stratified cases; these form a series of increasing expressive generality. However, OLP's have problematic semantics for the unrestricted case, due essentially to the interaction of recursion with negation-as-failure. "Recursion" here means that there is a *cyclic* (path of syntactic) dependency among the predicates (or, more generally, among the ground atoms) through rules. More precisely, a logic program $\mathcal{E}$'s *predicate dependency graph* $PDG_{\mathcal{E}}$ is defined as follows. The vertices of the graph are the predicates that appear in $\mathcal{E}$. $\langle p, q \rangle$ is a (directed) edge in $PDG_{\mathcal{E}}$ iff there is a rule r in $\mathcal{E}$ with p in its head and q in its body. "Predicate-acyclic" means that there are no cycles in the predicate dependency graph. "Stratified" (in its various flavors) means cycles of a restricted kind are allowed.

The well-founded semantics is probably the currently most popular semantics for the unrestricted case, and is our favorite. With WFS, the unrestricted case always has a single set of conclusions, and is tractable under commonly-met restrictions (e.g., VBD defined below).

Our approach for an initial practically-oriented LP-based business rules KR is, however, to keep to expressively restricted cases that have uncontroversial (i.e., consensus in the research community) semantics — starting with predicate-

[5] "Ordinary" logic programs are also sometimes known as "general" (e.g., in [1]) or "normal" logic programs, or as (the declarative version of) "pure Prolog".

acyclic. Compared to these uncontroversial cases, the unrestricted case (e.g., with WFS) is more complex computationally and, perhaps even more importantly, is more difficult in terms of software engineering. It requires more complicated algorithms and is not widely deployed. The predicate-acyclic expressive restriction can be checked syntactically with a relatively simple algorithm and with relatively low computational cost.

An OLP (with WFS) consists of a set of premise rules, and entails a set of (primitive) conclusions. In a predicate-acyclic OLP, each conclusion has the form of a ground atom. Intuitively, each conclusion atom is believed to be $true$, and every other (i.e., non-conclusion) ground atom is not believed to be $true$ (recall $\sim$ above).

Next, we discuss the advantages of OLP's (with WFS). **(Adv 1)**: OLP has a fully declarative semantics that is useful and well-understood theoretically. **(Adv 2)**: OLP includes negation-as-failure and thus supports basic logical non-monotonicity. **(Adv 3)**: OLP has considerable expressive power, yet is relatively simple and is not overkill expressively. **(Adv 4)**: OLP, unlike first-order-logic/KIF, is computationally tractable in the following sense. Under commonly met expressive restrictions, e.g., VBD, inferencing — i.e., rule-set execution — in OLP can be computed in worst-case polynomial-time. Here, we say that an LP is "*VBD*" when either (a) it is ground, or (b) it has no logical functions of non-zero arity (a.k.a. the *Datalog* restriction[6]) and it has a bounded number of logical variables appearing in each rule. By contrast, classical logic, e.g., first-order logic — and thus KIF, is co-NP-hard under the VBD restriction. **(Adv 5)**: We observe predicate-acyclic OLP's semantics is widely shared among many commercially important rule-based systems as an expressive subset of their behavior. Prolog is the most obvious family of such systems. Relational databases are another such family: many SQL view definitions (and relational algebra operations) are in effect OLP rules. OPS5-heritage production rule systems are less closely related because of their extensive use of procedural attachments, but insofar as they commonly do pure-belief predicate-acyclic inferencing, their semantics is closely related to forward-directed OLP's. Event-condition-action rules are somewhat similar in this regard to a simple form of production rules. Other systems sharing the semantics include many expert/knowledge-based systems and intelligent agent building tools; many of these implement forward-directed predicate-acyclic OLP's. Predicate-acyclic OLP semantics thus reflects a consensus in the rule representation community that goes beyond the logic programming community. **(Adv 6)**: Predicate-acyclic OLP's are, in effect, widely implemented and deployed, including in Prolog's and SQL relational databases, but also beyond them.

[6]It is usually straightforward to representationally reformulate a rule-set so as to replace a logical function f having arity $k > 0$ by a predicate fp having arity $k + 1$, where intuitively the $(k + 1)^{th}$ argument corresponds to the result of applying the function.

(Adv 7): There is a large population of developers (not just researchers) who are familiar with them.

4.2 KR Extension to Courteous Logic Programs

Courteous LP's (CLP) expressively extends Ordinary LP's (with WFS) by adding the capability to conveniently express prioritized conflict handling, while maintaining computational **tractability** (e.g., under the VBD restriction). CLP is a previous KR [10] which we have here further expressively generalized, notably to handle *pairwise mutual exclusion conflicts*, rather than simply conflicts of the form p versus $\neg p$. (Here, $\neg$ stands for classical negation. Intuitively, $\neg p$ means p is believed to be definitely $false$.)

CLP can be tractably compiled to OLP. Indeed, we have implemented such a *courteous compiler* [11] [13] [12]. The compiler enables modularity in software engineering and eases implementation and deployment: the Courteous LP expressive capability can be added modularly to an Ordinary LP rule engine/system simply by adding a pre-processor. Compilation's computational complexity is cubic, worst-case, but often is closer to linear.

In this paper, we do not have space to give full details about the generalized CLP formalism or its courteous compiler. Instead, here we will give an overview and some examples. For details, see [12] and the forthcoming extended version of this paper.

CLP handles conflicts between rules using partially-ordered prioritization information that is naturally available based on relative specificity, recency, and authority. Rules are subject to override by higher-priority conflicting rules. For example, some rules may be overridden by other rules that are special-case exceptions, more-recent updates, or from higher-authority sources.

Courteous LP's facilitate specifying sets of rules by merging, updating and accumulating, in a style closer (than Ordinary LP's or than classical logic/KIF) to natural language descriptions. The expressive extension provided by CLP is thus valuable especially because it greatly facilitates incremental specification, by often eliminating the need to explicitly modify previous rules when updating or merging. In terms of the rule representation requirements we gave in section 3, this enables significantly greater modularity, locality, ease of modification, expressive convenience, and conceptual naturalness.

In CLP, priorities are represented via a fact comparing rule labels: $overrides(rule1, rule2)$ means that $rule1$ has higher priority than $rule2$. If $rule1$ and $rule2$ conflict, then $rule1$ will win the conflict.

Syntactically, a CLP rule differs from an OLP rule in two ways. First, it may have an optional rule label, used as a handle for specifying prioritization. Second, each rule literal may be classically negated. Syntactically, OLP is simply a special case of CLP. An OLP rule lacks a label and does not mention classical negation.

A CLP rule has the form:

$$\langle lab \rangle \quad L_0 \;\leftarrow\; L_1 \wedge \ldots \wedge L_m \wedge \sim\! L_{m+1} \wedge \ldots \wedge \sim\! L_n.$$

Here, $n \geq m \geq 0$, and each L_i is a classical literal. A *classical literal* is a formula of the form A or $\neg A$, where A is an atom. $\neg$ stands for the *classical negation* operator symbol, and is read in English as "not". lab is the rule's label. The label is optional. If omitted, the label is said to be *empty*. The label is not required to be unique within the scope of the overall logic program; i.e., two rules may have the same label. The label is treated as a 0-ary function symbol. The label is preserved during instantiation; all the ground instances of the rule above have label lab. $overrides$ and the labels are treated as part of the language of the logic program, similarly to other predicate and function symbols appearing in the logic program.

Semantically, a CLP entails a set of (primitive) conclusions each of which is a ground classical literal. Classical negation is enforced: p and $\neg p$ are never both concluded, for any (classical literal) p. This can be viewed as the enforcing of an implicit mutual exclusion between p and $\neg p$.

In the newly generalized version of CLP, the CLP also optionally includes a set of pairwise mutual exclusion (*mutex*) statements, along with the rules. These mutex's specify the scope of conflict. An *unconditional* mutex has the syntactic form:

$$\perp \;\leftarrow\; L_1 \wedge L_2.$$

where each L_i is a classical literal. Semantically, each such mutual exclusion specified by a mutex is enforced: L_1 and L_2 are never both concluded (for any instantiation). These mutex's are particularly useful for specifying that at most one of a set of alternatives is to be permitted. E.g., in Example 1, it is straightforward to specify via a mutex that the refund percentage must be at most one of the values {90% , 100%}. E.g., in Example 2, it is straightforward to specify via 3 mutex's that the lead time must be at most one of the values {2 days, 14 days, 30 days}. E.g., in Example 3, it is straightforward to specify via 3 mutex's that the discount percentage must be at most one of the values {0%, 5%, 10%}.

It is expressively more convenient sometimes to use a more general form of mutex: a *conditional* mutex, which has the syntactic form:

$$\perp \;\leftarrow\; L_1 \wedge L_2 \;\mid\; (?Y \neq ?Z).$$

where $?Y$ and $?Z$ are logical variables that appear respectively in L_1 and L_2. E.g.,

$$\perp \;\leftarrow\; giveDiscount(?Cust, ?YPercent) \wedge giveDiscount(?Cust, ?ZPercent) \;\mid\; (?YPercent \neq ?ZPercent).$$

This conditional mutex enables one to specify with a single mutex statement (rather than with three or more unconditional mutex's) that for a given customer $?Cust$ there must be at most one value concluded for the discount percentage.

The enforcement of classical negation can be viewed as a set of *implicit* unconditional mutex's, one for each predicate Q, that each have the form

$$\perp \;\leftarrow\; Q(?X1, \ldots, ?Xm) \wedge \neg Q(?X1, \ldots, ?Xm).$$

where Q's arity is m. This is called a *classical* mutex.

Example 4 (Ordering Lead Time, in CLP)

Example 2 can be straightforwardly represented in CLP as follows:

$$\langle a \rangle \quad orderModificationNotice(?Order, days14) \;\leftarrow\; preferredCustomerOf(?Buyer, ?Seller) \wedge purchaseOrder(?Order, ?Buyer, ?Seller).$$

$$\langle b \rangle \quad orderModificationNotice(?Order, days30) \;\leftarrow\; minorPart(?Order) \wedge purchaseOrder(?Order, ?Buyer, ?Seller).$$

$$\langle c \rangle \quad orderModificationNotice(?Order, days2) \;\leftarrow\; preferredCustomerOf(?Buyer, ?Seller) \wedge orderModificationType(?Order, reduce) \wedge orderItemIsInBacklog(?Order) \wedge purchaseOrder(?Order, ?Buyer, ?Seller).$$

$$overrides(c, a).$$

$$\perp \;\leftarrow\; orderModificationNotice(?Order, ?X) \wedge orderModificationNotice(?Order, ?Y) \;\mid\; (?X \neq ?Y).$$

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 (recall the discussion of modularity and ease of modification in section 3).

Semantically, the prioritized conflict handling in CLP is defined by a process of **prioritized argumentation among opposing candidates**. Opposition is specified by the set of mutex's. Each rule r whose body is satisfied, i.e., which "fires", generates a candidate c for r's (appropriately instantiated) head p. This candidate has an associated label, which is simply that rule r's label. In general, there may be multiple candidates for a given p, i.e., a *team* for p. If and only if there is an opposing candidate d (i.e., a candidate for an opposing literal q) that has higher priority than candidate c, then c is *refuted*. Suppose there is an unrefuted candidate for p. If there are no unrefuted candidates for any opposer of p, then p *wins*, i.e., p is concluded. However, it may be that there is an unrefuted candidate for an opposer of p; in this case, the opposing unrefuted candidates *skeptically defeat* each other. The conflict cannot be resolved by the specified priority; neither p nor its opposer is concluded.

Another way to view this is as follows. An *opposition-locale* is a set of $h \geq 2$ ground classical literals that oppose each other, such that at most one of those literals is permitted (by the specified mutex's) to be concluded. In each opposition-locale, if the maximal-priority candidates are all for the same literal, then that literal wins.

The definition of CLP includes some additional expressive restrictions, which we do not have space to detail here.

CLP always produces a **consistent** set of conclusions, enforcing all the mutex's. CLP also has several other attractive well-behavior properties, including about merging and about natural behavior of prioritization; however, we do not have space here to detail these.

The courteous compiler in effect represents the prioritized argumentation process in OLP. It introduces some extra "*adorning*" predicates to represent the intermediate stages of argumentation: candidates, unrefuted candidates, skeptical defeat, etc.. The compiler makes it simple to detect an unresolved conflict, e.g., to raise an alarm about it in either forward or backward reasoning.

From the tractability of courteous compilation, it follows directly that Courteous LP inferencing under the VBD restriction is tractable. It has the same worst-case time and space complexity as: OLP inferencing where the bound v on the number of variables per rule has been increased to $v + 2$.

Note that CLP overlaps syntactically and semantically with KIF for a broad case. CLP without negation-as-failure, without labels (or ignoring labels), and without conflict, is syntactically a restricted (essentially, clausal) case of first-order-logic (FOL)/KIF. Semantically, such CLP is sound but incomplete when compared to FOL/KIF, in that its entailed conclusions are equivalent to a (conjunctive) set of ground literals.

4.3 XML Embodiment: Business Rules Markup Language

Our approach includes a second aspect beyond the fundamental KR. We embody the rule representation concretely as XML documents.

Next, we give, for the first time, an XML embodiment of CLP. Called *Business Rules Markup Language (BRML)*, this XML embodiment functions as an interlingua between heterogeneous rule representations/systems which different contracting agents may be employing. BRML inherits the declarative semantics of CLP.

BRML also is the first XML embodiment of (declarative) OLP to our knowledge. Since CLP also expressively covers a subset of KIF, as described above, BRML is also an XML embodiment of that subset of KIF — to our knowledge, the first XML embodiment of (any fragment of) KIF.

Figure 1 shows the CLP from Example 4 encoded in BRML. Only the first rule of that Example is shown in detail. "`cliteral`" means "classical literal". "`fcliteral`" means a rule body literal, i.e., a literal formed by optionally applying the negation-as-failure operator outside/in-front of a classical literal.

In this paper, we do not have space to give full details about the XML encoding. For full details, see: (1) the CommonRules prototype download package (at

```
<?xml version="1.0"?>
<clp>
 <erule rulelabel="a">
  <head>
   <cliteral predicate=
              "orderModificationNotice">
    <variable name="Order"/>
    <function name="days14"/>
   </cliteral>
  </head>
  <body>
   <and>
    <fcliteral predicate=
                "preferredCustomerOf">
     <variable name="Buyer"/>
     <variable name="Seller"/>
    </fcliteral>
    <fcliteral predicate="purchaseOrder">
     <variable name="Order"/>
     <variable name="Buyer"/>
     <variable name="Seller"/>
    </fcliteral>
   </and>
  </body>
 </erule>

  ... [rest of rules & mutex's skipped]
</clp>
```

Figure 1: XML, i.e., BRML, for Example 4.

`http://alphaworks.ibm.com`), which contains the XML Data Type Definition (DTD) for BRML, explanation of it, and a number of examples (esp. "orderingleadtime" there); and (2) the forthcoming extended version of this paper. The current DTD is in draft form; updates to it will be posted on the authors' websites. See [14] for how BRML fits into the larger context of agent communication languages, including the FIPA Agent Communication Language (ACL) draft standard.

As compared to the usual plain ASCII text style of embodiment cf. KIF or Prolog or most programming languages, the XML approach has several advantages. It facilitates developing/maintaining parsers (via standard XML parsing tools), and integrating with WWW-world software engineering. XML is easier to automatically parse, generate, edit, and translate, because there are standard XML-world tools for these tasks. The hyper-text (i.e., links) aspects of XML are also useful. For example, a rule set may via XML have some associated URL's which point to documents describing that rule set's knowledge representation or authors or application context. Or it may have associated URL's which point to tools for processing that rule set, e.g., to execute it, edit it, analyze it, or validate it (syntactically or semantically). Particularly useful for our nearer-term purposes is that an associated URL may point to documents describing the semantics and algorithms for translator services or components, as well as to translator tools and examples. Representing business rules in XML has a further advantage: it will complement domain-specific ontologies (i.e., vocabularies) available in XML. Many such ontologies exist already, and many more are expected to be developed in the next few years, including in e-commerce domains. The XML approach also facilitates integration with Electronic Data Interchange (EDI) and other e-commerce components that "talk" XML.

We have implemented sample translators that go (bidirectionally) from the XML interlingua to several actual rule systems as proof of feasibility. These rule systems include two previously existing WFS OLP inferencing engines built by others and implemented in C. One is exhaustive forward-direction: Smodels (version 1), by Ilkka Niemela and Patrik Simons, `http://saturn.hut.fi/html/staff/ilkka.html`. The other is backward-direction: XSB, by David Warren *et al*, `http://www.cs.sunysb.edu/~sbprolog`. In addition, we have implemented a sample translator to a third, predicate-acyclic WFS OLP inferencing engine we built ourselves in Java. Furthermore, we have implemented a sample translator to ANSI-draft KIF (ASCII format).

5 Discussion and Future Work

The **implementation** of our approach was released as a free alpha prototype called **CommonRules** on the Web in July of 1999, at `http://alphaworks.ibm.com`. This prototype is a Java library that includes both the CLP and the BRML aspects of our approach. In particular, it includes a courteous compiler and sample translators between BRML and several other rule systems/languages.

In summary, we believe our approach combining CLP and XML meets the whole set of requirements we gave in section 3 better than any previous approach. Indeed, we believe our approach meets every one of those requirements to a significant degree, with one exception: the ability to express procedural attachments.

The usefulness of rules in a declarative KR for representing executable specifications of contract agreements is based largely on their following advantages relative to other software specification approaches and programming languages. First, rules are at a relatively high level of abstraction, closer to human understandability, especially by business domain experts who are typically non-programmers. Second, rules are relatively easy to modify dynamically and by such non-programmers.

In current work, we are expressively generalizing further to *Situated* Courteous LP's, so as to enable procedural attachments as well — in a semantically clean manner (i.e., declaratively in a particular well-defined sense). Situated LP's[9] [16] [15], another expressive extension of Ordinary LP's, hook beliefs to drive procedural APIs. Procedural attachments for condition-testing ("*sensing*") and action-performing ("*effecting*") are specified as part of the knowledge representation: via sensor and effector *link* statements. Each sensor or effector link associates a predicate with an attached procedure.[7]

In current work, we are also expressively generalizing CLP and BRML further to relax expressive restrictions such as on cyclicity/recursion restriction, e.g., to be stratified rather than predicate-acyclic.

There are several other formalisms for prioritized LP's that have similar syntax to Courteous LP's (except for lacking mutex's) but different semantics in regard to conflict handling (see [10] [11] for a review). A direction in our current work is to explore this dimension of heterogeneity. None of these other formalisms to our knowledge has as attractive a combination of useful expressive power, software-engineering modularity, well-behavior (e.g., consistency, unique set of conclusions), tractability, and conceptual simplicity (e.g., in prioritization behavior and merging). In particular, none can express mutex's (besides implicit classical mutex's), and none has a compiler to OLP's.

There are other, more expressively powerful approaches to prioritized default reasoning such as Prioritized Default Logic [2] and Prioritized Circumscription [19] [18] [7] [8] that essentially can express mutex's, but in these the prioritized conflict handling imposes computationally intractable overhead [6].

It appears fairly straightforward to extend our BRML DTD in stages so as to express full first-order logic and then full KIF. A direction for future work is to create a DTD, maximally compatibly with BRML, that expresses full KIF.

In other work [21], we have extended our contract rule representation approach with negotiation features oriented towards automatic configuration of auctions, including to specify which attributes of a contract are to be the subject of negotiation or bidding.

Of course, there is yet more to do to fulfill our approach's promise, and achieve its ultimate goals. Further issues for future work include: meshing more closely with other aspects of contracts, e.g., transactions, payments, negotiation and communication protocols [5] [20], EDI, and utility/cost-benefit; fleshing out the relationships to a variety of commercially important rule representations/systems; representing constraints as in constraint satisfaction and Constraint Logic Programs; and representing delegation as in security/authorization policies [17].

An extended version of this paper will be available as a forthcoming IBM Research Report (at `http://www.research.ibm.com`).

Acknowledgements

Michael Travers (while at IBM T.J. Watson Research Center) and Xiaocheng Luan (of UMBC, while at IBM T.J. Watson Research Center), contributed to the current implementation of the CLP KR, BRML, and the associated translators.

[7]Note that "link" here does not mean in the sense of an XML or HTML hypertext link.

Michael Travers' contribution was especially to the XML embodiment. Miao Jin (UMBC) contributed to the XML DTD. Daniel Reeves and Michael Wellman (of U. Michigan) contributed to the formulation of the overall context and motivation for automated contract languages. Thanks to anonymous reviewers for editing suggestions.

References

[1] Chitta Baral and Michael Gelfond. Logic programming and knowledge representation. *Journal of Logic Programming*, 19,20:73–148, 1994.

[2] Gerhard Brewka. Reasoning about priorities in default logic. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pages 940–945, Menlo Park, CA / Cambridge, MA, 1994. AAAI Press / MIT Press.

[3] W. F. Clocksin and C.S. Mellish. *Programming in Prolog*. Springer-Verlag, 1981.

[4] Thomas Cooper and Nancy Wogrin. *Rule-based Programming with OPS5*. Morgan Kaufmann Publishers, San Francisco, CA, 1988. ISBN 0-934613-51-6.

[5] Asit Dan, D. Dias, T. Nguyen, M. Sachs, H. Shaikh, R. King, and S. Duri. The Coyote Project: Framework for Multi-party E-Commerce. In *Proc. 7th Delos Workshop on Electronic Commerce. Lecture Notes in Computer Science, Vol. 1513.* Springer-Verlag, 1998.

[6] Georg Gottlob. Complexity results for nonmonotonic logics. *Journal of Logic and Computation*, 2:397–425, 1992.

[7] Benjamin N. Grosof. Generalizing Prioritization. In *Proc. 2nd Intl. Conf. on Principles of Knowledge Representation and Reasoning (KR-91)*, 1991.

[8] Benjamin N. Grosof. *Updating and Structure in Non-Monotonic Theories*. PhD thesis, Computer Science Dept., Stanford University, Oct. 1992.

[9] Benjamin N. Grosof. Building Commercial Agents: An IBM Research Perspective (Invited Talk). In *Proc. 2nd Intl. Conference and Exhibition on Practical Applications of Intelligent Agents and Multi-Agent Technology (PAAM97)*, P.O. Box 137, Blackpool, Lancashire, FY2 9UN, UK.

[10] Benjamin N. Grosof. Prioritized conflict handling for logic programs. In Jan Maluszynski, editor, *Logic Programming: Proceedings of the International Symposium (ILPS-97)*, MIT Press, 1997.

[11] Benjamin N. Grosof. Compiling Prioritized Default Rules Into Ordinary Logic Programs. IBM Research Report RC 21472.

[12] 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/people/g/grosof/papers.html, July 1999. Included in CommonRules documentation released July 30, 1999 at http://alphaworks.ibm.com . Revised version forthcoming as IBM Research Report.

[13] Benjamin N. Grosof. DIPLOMAT: Compiling Prioritized Default Rules Into Ordinary Logic Programs, for E-Commerce Applications (extended abstract of Intelligent Systems Demonstration). In *Proc. AAAI-99*, 1999. Morgan Kaufmann. Extended version is IBM Research Report RC 21473.

[14] Benjamin N. Grosof and Yannis Labrou. An Approach to using XML and a Rule-based Content Language with an Agent Communication Language. In *Proc. IJCAI-99 Workshop on Agent Communication Languages*, 1999.

[15] Benjamin N. Grosof, David W. Levine, and Hoi Y. Chan. Flexible procedural attachments to situate reasoning systems. In *U.S. Patent 5,778,150 (July 7, 1998)*, Washington, D.C., USA

[16] Benjamin N. Grosof, David W. Levine, Hoi Y. Chan, Colin P. Parris, and Joshua S. Auerbach. Reusable Architecture for Embedding Rule-Based Intelligence in Information Agents. In *Proc. ACM Conf. on Information and Knowledge Management (CIKM-95) Workshop on Intelligent Information Agents*, 1995.

[17] Ninghui Li, Joan Feigenbaum, and Benjamin N. Grosof. A logic-based knowledge representation for authorization with delegation (extended abstract). In *Proc. 12th Intl. IEEE Computer Security Foundations Workshop*, 1999. Extended version is IBM Research Report RC 21492.

[18] V. Lifschitz. Computing circumscription. In *Proceedings Intl. Joint Conf. on AI (IJCAI-85)*, 1985.

[19] J. McCarthy. Applications of circumscription to formalizing commonsense knowledge. *Artificial Intelligence*, 28:89–116, 1986.

[20] Naftaly H. Minsky and Victoria Ungureanu. A Mechanism for Establishing Policies for Electronic Commerce. In *Proc. 18th Intl. Conf. on Distributed Computing Systems (ICDCS)*, May 1998.

[21] Daniel M. Reeves, Benjamin N. Grosof, Michael Wellman, and Hoi Y. Chan. Toward a Declarative Language for Negotiating Executable Contracts. In *Proc. AAAI-99 Workshop on Artificial Intelligence in Electronic Commerce (AIEC-99)*.

[22] Jeffrey D. Ullman and Jennifer Widom. *A First Course in Database Systems*. Prentice-Hall, 1997.

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