RC 21476 (96914) 11 May 1999
Computer Science

IBM Research Report

Toward a Declarative Language for Negotiating Executable Contracts

Daniel M. Reeves, Benjamin N. Grosof, Michael P. Wellman, and Hoi Y. Chan

University of Michigan Artificial Intelligence Laboratory
1101 Beal Avenue, Ann Arbor, MI 48109-2110 USA
Internet e-mail: {dreeves,wellman}@umich.edu
Web: http://ai.eecs.umich.edu/people/{dreeves,wellman}

IBM Research Division
T.J. Watson Research Center
P.O. Box 704, Yorktown Heights, NY 10598, USA
(914) 784-7100
Internet e-mail: {grosof,hychan}@us.ibm.com (alt. grosof@cs.stanford.edu)
Web: http://www.research.ibm.com/people/g/grosof

IBM Research Division
Almaden · Austin · Beijing · Delhi · Haifa · T.J. Watson · Tokyo · Zurich

Abstract: We give an approach to automating the negotiation of business contracts. Our goal is to develop a language for both (1.) fully-specified, executable contracts and (2.) partially-specified contracts that are in the midst of being negotiated, including via automated auctions. Our starting point for this language is Courteous Logic Programs (CLP's), a form of logic-based knowledge representation (KR) that is semantically declarative, intuitively natural, computationally tractable, and practically executable. A CLP is suitable in particular to represent a fully-specified executable contract. The basic CLP KR also facilitates modification during negotiation, because it includes prioritized conflict handling features that facilitate modification. Beyond the basic CLP KR, we have developed an initial ontology, and an associated style of representation, to specify additional aspects of a partial contract and of a negotiation process. The initial ontology specifies the set of negotiables and the structure of a contract in terms of its component goods/services and attributes. Specifying the negotiable aspects of a good or service includes specifying its attributes, their possible values, and dependencies/constraints on those attributes. Building upon the representation of these negotiable aspects, we are in current work developing methods to structure negotiations, especially to select and configure auction mechanisms to carry out the negotiation. This work brings together two strands of our previous work on business process automation in electronic commerce: representing business rules shared between enterprises, and configurable auction mechanisms.

Related Papers and Material: can be found via the authors' Web addresses and as IBM Research Reports at http://www.research.ibm.com .

Keywords: contracts, negotiation, auctions, e-commerce, electronic commerce, specification, logic programs, default reasoning, rules, priorities, non-monotonic reasoning, intelligent agents.

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. The overall contracting process comprises several stages, including broadly:

1. *Discovery.* Agents find potential contracting partners.
2. *Negotiation.* Contract terms are determined through a communication process.
3. *Execution.* Transactions and other contract provisions are executed.

In this work we are concerned primarily with negotiation, and specifically with the process by which an automated negotiation mechanism can be configured to support a particular contracting episode. Our goal is a shared language with which agents can define the scope and content of a negotiation, and reach a common understanding of the negotiation rules and the contract implications of negotiation actions. Note that we make a sharp distinction between the definition of the negotiation mechanism, and the actual negotiation strategies to be employed by participating agents. Our concern here is with the former, though of course in designing a mechanism one must consider the private evaluation and decision making performed by each of the negotiating parties.

2 Overview of Problem and Approach

The central question in configuring a contract negotiation is "What is to be negotiated?" In any contracting context, some features of the potential contract must be regarded as fixed, with others to be determined through the contracting process. At one extreme, the contract is fully specified, except for a single issue, such as price. In that case, the negotiation can be implemented using simple auction mechanisms of the sort one sees for specified goods on the Internet. The other extreme, where nothing is fixed, is too ill-structured to consider automating to a useful degree in the current state of the art.

Most contracting contexts lie somewhere in between, where an identifiable set of issues are to be determined through negotiation. Naturally, there is a tradeoff between flexibility in considering issues negotiable and complexity of the negotiation process. But regardless of how this tradeoff is resolved, we require a means to specify these issues, so that we can automatically configure the negotiation mechanisms that will resolve them. That is, we require a *contracting language*—a medium for expressing the contract terms resulting from a negotiation.

2.1 Contracting Language

In developing a shared contracting language, we are concerned with the three stages of contracting: discovery, negotiation, and execution. This multiplicity of purpose is one argument for adopting a declarative approach, with a relatively expressive knowledge representation (KR). "Declarative" here means that the semantics say which conclusions are entailed by a given set of premises, without dependence on procedural or control aspects of inference algorithms. In addition to flexibility, such an approach promotes standardization and human understandability.

Traditionally, of course, contracts are specified in legally enforceable natural language ("legalese"), as in a typical mortgage agreement. This has great expressive power—but often, correspondingly great ambiguity, and is thus very difficult to automate.[1], and is thus very difficult to automate. At the other extreme are automated languages for restricted domains; in these, most of the meaning is implicit in the automated representation. This is the current state of Electronic Data Interchange (EDI). We are in the sparsely occupied middle ground, aiming for considerable expressive power but also considerable automatability.

Our point of departure for our KR is pure logic programs (in the knowledge-representation-theory sense, not Prolog). (Baral & Gelfond [1] provide a helpful review.) Logic programs are not only declarative and relatively powerful expressively, but also practical, relatively computationally efficient, and widely deployed.

We embody the representation concretely as XML messages. This choice enhances human readability (via standard XML rendering/UI tools) and supports inclusion and generation of textual information. It also facilitates integration with EDI components. The XML approach further facilitates developing/maintaining parsers (via standard XML parsing tools), integrating with WWW-world software engineering, and the enriching capability to (hyper-)link to ontologies and other extra information. See [8] for details about the XML representation, its advantages, and its relationship to overall inter-agent communication.

Our KR builds on our prior work representing business rules in Courteous Logic Programs (CLP's) [3] [4] [6] [7] (see also Section "Courteous Logic Programs as KR"). To express executable contracts, these rules must specify the goods and services to be provided, along with applicable terms and conditions. Such terms include customer service agreements, delivery schedules, conditions for returns, usage restrictions, and other issues relevant to the good or service provided.

As part of our approach, we extend this KR with features specific to negotiation. Foremost among these is the ability to specify *partial* agreements, with associated negotiable parameters. A partial agreement can be viewed as a contract template. Some of its parameters may be bound to particular values while others may be left open.

2.2 Negotiable Parameters

Once we have this contracting language, our next step will be to use it to establish the automated negotiation process. As noted above, a key element of this is to identify the negotiable parameters. The contract template effectively defines these parameters by specifying what the contract will be for any instantiation of parameter values.

The problem then, is to enable the contract language to allow descriptions of contract templates. In addition, we require auxiliary specification of possible values for parameters, and dependencies and constraints among them. Given this specification of what can be negotiated, we require a policy to determine what is actually to be included in the given

[1]Even if a natural language contract is completely unambiguous, it would require a vast amount of background and domain knowledge to automate.

negotiation episode (rather than assigned a default value, or left open for subsequent resolution).

This answers the question of *what* is to be negotiated; the remaining question is *how*. In general, there are many ways to structure a negotiation process to resolve multiple parameters. We focus on processes mediated by *auctions*. As we describe below, the problem then becomes one of *configuring* appropriate auctions to manage the negotiation.

3 Auction-Based Negotiation

Mechanisms for determining price and other terms of an exchange are called *auctions*. Although the most familiar auction types resolve only price, it is possible to define *multidimensional* generalizations and variants that resolve multiple issues at once. This can range from the simple approach of running independent one-dimensional auctions for all of the parameters of interest, to more complicated approaches that directly manage higher-order interactions among the parameters.

Auctions are rapidly proliferating on the Internet.[2] Although typical online auctions support simple negotiation services, researchers have begun to deploy mechanisms with advanced features. For example, our own Michigan Internet AuctionBot supports a high degree of configurability [11] (http://auction.eecs.umich.edu/), and IBM's auction system supports one-sided sales auctions integrated with other commerce facilities [9].

Although multidimensional mechanisms are more complicated, and not yet widely available, we expect that they will eventually provide an important medium for automated negotiation. For example, *combinatorial* auctions allow bidders to express offers for combinations of goods, and determines an allocation that attempts to maximize overall revenue. We are aware of one prototype system currently supporting combinatorial auctions over the Internet [10]. *Multiattribute* auctions, typically employed in procurement, allow specification of offers referring to multiple attributes of a single good [2].

Whether a multiattribute auction, a combinatorial auction, or an array of one- or zero-dimensional auctions is appropriate depends on several factors. Although a full discussion is beyond the scope of this paper, we observe that these factors can bear on any of:

- The *legality* of auction configurations. For example, if some attributes are inseparable (i.e., both must be specified in the contract), then it makes no sense to treat them as separate goods in a combinatorial auction.

- The *expected performance* of auction configurations. For example, if parameters represent distinct and separable contract options, then they could be handled either by separate or combined auctions. Whether they should be combined depends on how *complementary* they are as perceived by the negotiating agents.

- The *complexity* of auction configurations, for both the mechanism infrastructure and participating agents. Dimensionality plays a large role in complexity tradeoffs.

[2] Looking at Yahoo alone yields 104 auction services listed, and 120,000 active auctions on their own service (http://auctions.yahoo.com/).

4 Our Approach

4.1 Courteous Logic Programs as KR

Next, we discuss our approach to the fundamental KR used for describing contract agreements.

Rules as an overall representational approach capture well many aspects of what one would like to describe in automated contracts. Rules are useful generally to represent much of the substantive contents of negotiation messages, especially to describe products and services that are offered or requested. This includes, for example: offers, bids, and proposals; requests for bids or proposals; requests for quotations (RFQs); and surrounding agreements such as contractual terms and conditions, and customer service agreements. Rules are also useful to represent relevant aspects of business processes, e.g., how to place an order, return an item, or cancel a delivery.

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.

Our point of departure is a particular form of rules: *predicate-acyclic pure-belief* logic programs (LP's). Here, we mean "logic programs" in the sense of pure-belief knowledge representation, rather than in the sense of the Prolog programming language. "Pure-belief" here means without procedural attachments. "Predicate-acyclic" means without cyclic/recursive paths of dependence among the rules' predicates.[3]

This KR has a deep semantics that is useful, well-understood theoretically, and highly declarative. This semantics reflects a consensus in the rules representation community; it is widely shared among many commercially important rule-based systems and relational database systems. This core is also relatively computationally efficient.[4]

Logic programs are relatively simple and are not overkill representationally. Logic programs are also relatively fast computationally. Under commonly met restrictions (e.g., no logical functions of non-zero arity, a bounded number of logical variables per rule), inferencing — i.e., rule-set execution — in LP's can be computed in worst-case polynomial-time.[5]

The KR we are using to represent contracts is **Courteous** Logic Programs. Courteous LP's expressively generalize the *ordinary* LP's (described above) by adding the capability to conveniently express prioritized conflict handling, i.e., where some rules are subject to override by higher-priority conflicting rules. For example, some rules may be overridden by

[3]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 $\mathcal{E}$. $\langle p_i, p_j \rangle$ is a (directed) edge in $PDG_{\mathcal{E}}$ iff there is a rule r in $\mathcal{E}$ with p_i in its head (i.e., consequent) and p_j in its body (i.e., antecedent). "Predicate-acyclic" means that there are no cycles in the predicate dependency graph.

[4]The general case of LP's, with unrestricted recursion/cyclicity interacting with negation-as-failure, has problems semantically, 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.

[5]Unlike classical logic, e.g., first-order logic, which is NP-complete under these restrictions, and semi-decidable without these restrictions

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 and updating and accumulation, in a style closer (than ordinary LP's) to natural language descriptions.

Courteous LP's include priorities, between rules, that are partially-ordered. Classical negation is enforced: p and classical-negation-of-p are never both concluded, for any belief expression p. 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.

The version of Courteous LP's we are using, partially described in [7] and [6], is further expressively generalized as compared to the previous version in [5] and [4].

Example: Modification Lead-Time

The English description of a business-to-consumer electronic commerce preferred-customer draft contract communicated from a airline (seller) to a traveler (buyer) might include a contract clause that comprises the following two business rules. Described in English, the first rule is:

```
Buyer can modify the departure time up
until 14 days before scheduled
departure, if
- the buyer is a preferred customer.
```

The second rule is:

```
Buyer can modify the departure time of
an item up until 2 days before
scheduled departure, if
- the buyer is a preferred customer, and
- the modification is to postpone the
  departure, and
- the current flight is full.
```

This second rule is a special-case rule and overrides the more general-case rule. (The rationale is that when the current flight is full the airline has demand for extra seats.)

These rules are straightforwardly represented in Courteous LP's, e.g., as:

```
<leadTimeRule1>
modificationNotice(?Buyer, ?Seller,
                   ?Flight, 14days) <-
  preferredCustomerOf(?Buyer, ?Seller).

<leadTimeRule2>
modificationNotice(?Buyer, ?Seller,
                   ?Flight, 2days) <-
  preferredCustomerOf(?Buyer, ?Seller) AND
  modificationType(?Flight, postpone) AND
  flightIsFull(?Flight).
```

```
overrides(leadTimeRule2, leadTimeRule1)
```

Here the arrow ("`<-`") indicates "if" and the "`?`" prefix indicates a logical variable.

Courteous LP's have several virtues semantically and computationally. A Courteous LP is guaranteed to have a consistent, as well as unique, set of conclusions. Priorities and merging behave in an intuitively natural fashion. Execution (inferencing) of courteous LP's is fast: only relatively low computational overhead is imposed by the conflict handling.[6]

Our work on representing contracts via courteous LP's builds on our prior work on representing business rules via courteous LP's (see http://www.research.ibm.com/people/g/grosof). We have a **running prototype implementation** [7] of Courteous LP's as a Java library, **including XML** formatting, rule specification, and rule inferencing/execution. An initial version of the prototype will be released as a free Web alpha in the spring of 1999.

4.2 Ontology for Specifying Partial Contracts

At an abstract level, what distinguishes a contract template from a fully-specified contract is that the contract template contains a set of variables, and the goal of the negotiation is to find an assignment to those variables. Once the variables are bound to specific values, there is a fully-specified contract. We call these variables the *negotiable parameters* (or *negotiable attributes*). To support performing this negotiation, the language of the contract must express the appropriate value ranges for, and constraints upon, the negotiable parameters.

We have talked about CLP as a basic KR suitable for specifying (via rules) an executable agreement. Beyond the basic KR we provide negotiation-specific ontology for expressing partially specified contracts and guiding and constraining the negotiation process. Below we give an initial set of such negotiation-level predicates.[7]

The first predicates we introduce allow bundling of attributes. The predicate `attribute(?Parent, ?Child)` allows us to create a tree of attribute bundles. If specified with the `attribute` predicate, the bundle of attributes is considered non-separable, i.e., it is not possible for a buyer to get some of the attributes from one seller and some from another. When it *is* possible to separate sets of attributes in this way[8], we use the predicate `component(?Parent, ?Child)` which again is used to impose an arbitrary tree structure of components and subcomponents on the negotiable attributes.

The `attribute` and `component` predicates are used to impose a hierarchy on negotiable parameters in the contract. Only the leaves of this tree structure may actually be negotiated, and this is indicated explicitly in our ontology with the predicate `negotiable(?NameOf-`

[6]For a previous version of courteous LP's, [6] gives the computational complexity analysis. The computational complexity of the further expressively generalized version is similar.

[7]They happen to all be predicates currently. In more extended versions of this approach we might find it useful to add logical functions as well.

[8]Although it is still up to the negotiation mechanism to determine whether or not components are actually supplied by different sellers.

NegotiablePredicate). This predicate indicates that the named predicate[9] represents a negotiable parameter of the contract.

Some parameters may be "negotiable" only in the sense that one party determines them and they are not open to counter-offers. We refer to these as *internal parameters*. Since these parameters are determined in the negotiation phase just like every other, we do not want to treat them specially in our ontology for representing negotiable aspects of the contract. Instead we introduce a special predicate, `negotiationType(?PredicateName, ?TypeOfNegotiation)`, where the second argument can take values such as `sellerChooses` or `buyerChooses`. It is straightforward in CLP to specify a default, "open for discussion."

The power of the negotiation-level predicates above is that they can be fully integrated into the existing framework of CLP. For example, we can specify that an attribute of the contract is only negotiable under certain conditions, or that the negotiation type depends on several factors including results of other negotiation. Results of other negotiations are easily reasoned about because they are simply facts in the rule set, such as `buyer(alice)` or `price(17)`.

Using the negotiation-level predicates presented, we now show the overall process for transforming a partial contract (or contract template) into a fully executable contract. A contract template consists of rules whose execution will fulfill the agreement (see Section "Courteous Logic Programs as KR"), a set of negotiable attributes (predicates whose names appear as arguments of `negotiable`), and rules about these attributes (those involving the negotiation-level predicates above as well as rules which have negotiable predicates as the head). First, the list of negotiable attributes is fed to the negotiation mechanism (considered a black box at this stage). Also feeding to the negotiation mechanism is the tree structure implied by the `attribute` and `component` rules. Additionally, the negotiation mechanism needs the results of inferencing from the rules about negotiable attributes (possibly it will need the rules themselves as well, i.e., the premises of that inferencing). This specifies constraints and dependencies among attributes.

When the negotiation mechanism completes, its output will be an assignment to all of the negotiable attributes. These will be represented as facts (recall that a negotiable attribute is simply a predicate whose name correspond to the attribute itself and whose argument is the value assigned to that attribute). When these facts are added to the original rule set (the partial contract) the contract will be fully executable.[10]

4.3 Examples

Here we present some example negotiation rules in the domain of travel packages to demonstrate the representation we are using. Note that these examples are meant to be illustrative of the expressiveness and flexibility of our representation, not as examples of how actual travel contracts should be specified.

Consider a contract for the purchase of a flight and hotel. The first thing we would like the partial contract to express is that the flight and the hotel are separable components—a

[9]This is currently restricted to unary predicates of the form `attribute(?Value)` but we may lift this restriction in the future to allow attributes that can be assigned tuples.

[10]These facts must be added at high priority (see 4.1) to ensure that they override any default values or constraints.

single buyer will not *necessarily* get both from the same seller. Each component has some (non-separable) attributes, yielding the following simple hierarchy:

```
component(contract, flight).
  attribute(flight, airline).
  attribute(flight, stopovers).
  attribute(flight, seatClass).
component(contract, hotel).
  attribute(hotel, quality).
```

The flight has various attributes, such as which airline (e.g., Northwest, Transworld, or American Airlines) and the number of stop-overs. An executable CLP contract would express such information with rules like the following:

```
flight(?Airline, ?FromCity, ?ToCity,
       ?Stopovers) <-
  airline(?Airline)
  AND stopovers(?Stopovers)
  AND possibleRoute(?Airline,
                    ?FromCity, ?ToCity).
```

To specify that certain attributes are negotiable, we use the predicate `negotiable` which takes the name[11] of a predicate from the contract as an argument:

```
negotiable('airline).
negotiable('stopovers).
```

If hotel cost were a parameter to the contract determined solely by the seller, this could be specified with the `negotiationType` predicate:

```
negotiable('hotelCost).
negotiationType('hotelCost, sellerChooses).
```

By definition, every subcomponent in the contract must have a price attribute[12], but this need not always be a negotiable parameter in the contract. For example, the total price of the travel package may be determined based on the negotiated values of flight price (adjusted by choice of seat class), hotel price, and discount:

```
flightPrice(?X) <-
  flightBasePrice(?BP) AND
  seatClassPrice(?SCP) AND
```

[11] We specify the name of the predicate rather than the predicate itself to avoid second-order logic. The quoting syntax used here is similar to Knowledge Interchange Format (KIF) quoting (see http://www.cs.umbc.edu/KIF).

[12] Price and quantity will remain distinguished by the mechanism since they are used in the scoring algorithm for multiattribute auctions [2].

```
  discount(?D) AND
  ?X == (1 - ?D) * (?BP + ?SCP).
price(?X) <- flightPrice(?FP) AND
  hotelCost(?HC) AND
  quantity(Q) AND X == Q * (?FP + ?HC).
```

4.4 Adding Negotiation Constructs to Existing Contracts

One important aspect of a contract template that does not at first appear to lend itself to our method of breaking down the template into a set of attributes with possible values, is the negotiation of what clauses to adopt or which criteria in the body of a given rule should actually be adopted. To capture this form of negotiation within our framework, we use boolean parameters to specify the adoption of rules and conjuncts/disjuncts as follows:

For a rule:

```
ruleHead <- ruleBody
  AND isRuleIncluded(yes).
negotiable('isRuleIncluded).
```

Note that when the negotiation mechanism completes it will add to the above rules exactly one of the following:

```
isRuleIncluded(yes).
isRuleIncluded(no).
```

For a conjunct:

```
... (conj OR isConjIncluded(no)) ...
negotiable('isConjIncluded).
```

For a disjunct:

```
... (disj AND isDisjIncluded(yes)) ...
negotiable('isDisjIncluded).
```

For example, consider the rule from Section "Courteous Logic Programs as KR" that the buyer can (conditionally) modify its order up until 2 days before scheduled delivery:

```
odificationNotice(?Buyer, ?Seller,
                  ?Flight, 2days) <-
  preferredCustomerOf(?Buyer, ?Seller) AND
  modificationType(?Flight, postpone) AND
  flightIsFull(?Flight).
```

For our mechanism to support negotiating the form of this rule (adoption of the rule itself and adoption of the two conjuncts), we modify it as follows:

```
modificationNotice(?Buyer, ?Seller,
                   ?Flight, 2days) <-
  isRuleIncluded(yes) AND
  (preferredCustomerOf(?Buyer, ?Seller)
   OR isPreferredCustomerRequired(no)) AND
  (modificationType(?Flight, postpone)
   OR isPostponeRequired(no)) AND
  flightIsFull(?Flight).
negotiable('isRuleIncluded).
negotiable('isPreferredCustomerRequired).
negotiable('isPostponeRequired).
```

Also, the above example included two constants (`2days` and `reduce`) which could be made negotiable by changing the constants to logical variables (e.g., `NoticeAmt` and `Type`), adding unary predicates (`noticeAmt` and `modificationType`), and making those predicates negotiable. In general,

```
foo(constant1, constant2) <- conditions.
```

would become

```
foo(?Var1, ?Var2) <- conditions AND
  var1(?Var1) AND var2(?Var2).
negotiable('var1).
negotiable('var2).
```

5 Discussion and Future Work

We have presented our approach of using a rule-based contract description language to specify negotiable parameters in a contract and discussed our planned approach for translating such a contract template into a set of auctions. It is worth mentioning that this work differs from existing work under similar names. Notably, Tuomas Sandholm's Contract Net and other work in distributed AI and industrial engineering describe mechanisms for subcontracting among agents in order to divide work in accomplishing a task. By contrast, our approach is to support an automated negotiation mechanism for agents to decide upon agreeable terms of a contract, which can then be executed electronically.

Another area that we will be working on, when looking at aspects of execution/enforcement of negotiated contracts, will be to link more closely with the procedures that will be performed as part of such execution/enforcement. For that purpose, it is desirable for the KR to conveniently express **"procedural attachments"**: the association of procedure calls (e.g., a call to a Java method ProcurementAuthorization.setApprovalLevel) with belief expressions (e.g., a logical predicate such as approvalAuthorizationLevel). We will thus expressively generalize further to **Situated** Courteous LP's. Situated logic programs [3] hook beliefs to drive procedural APIs. More precisely, situated LP's permit two semantically-clean kinds of procedural attachments for condition-testing ("sensing") and action-performing ("effecting"). Later we will also want to take a further step of expressive generalization to relax the cyclicity/recursion prohibition.

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] Fernando Branco. The design of multidimensional auctions. *Rand Journal of Economics*, 28:63–81, 1997.

[3] 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 .

[4] 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 [5].

[5] 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 .

[6] 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 Reserach Report RC 21472.

[7] Benjamin N. Grosof. DIPLOMAT: Compiling Prioritized Default Rules Into Ordinary Logic Programs, for E-Commerce Applications (extended abstract of Intelligent Systems Demonstration). In *Proceedings of AAAI-99*, San Francisco, CA, USA, 1999. Morgan Kaufmann. Extended version available in May 1999 as an IBM Research Report RC21473, http://www.research.ibm.com, search for Research Reports; P.O. Box 704, Yorktown Heights, NY 10598, USA.

[8] Benjamin N. Grosof and Yannis Labrou. An Approach to using XML and a Rule-based Content Language with an Agent Communication Language. In *Proceedings of the IJCAI-99 Workshop on Agent Communication Languages*, 1999. Held in conjunction with the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99) http://www.ijcai.org . Extended version available in May 1999 as IBM Research Report, http://www.research.ibm.com, search for Research Reports; P.O. Box 704, Yorktown Heights, NY 10598, USA.

[9] Manoj Kumar and Stuart I. Feldman. Internet auctions. In *Third USENIX Workshop on Electronic Commerce*, pages 49–60, Boston, 1998.

[10] Tuomas Sandholm. Approaches to winner determination in combinatorial auctions. *Decision Support Systems*, to appear.

[11] Peter R. Wurman, Michael P. Wellman, and William E. Walsh. The Michigan Internet AuctionBot: A configurable auction server for human and software agents. In *Second International Conference on Autonomous Agents*, pages 301–308, Minneapolis, 1998.