

Controlling Supplier Selection in an Automated Purchasing System

Pedro Szekely, Bob Neches, David P. Benjamin, Jinbo Chen, and Craig Milo Rogers

USC/Information Institute
Marina del Rey, CA 90292
(szekely, neches, benjamin, jinbo, rogers)@isi.edu
(310) 822-1511

Abstract

We present a system called DEALMAKER that allows users to specify policies that control selection among preferred suppliers in an automated purchasing system. The system gives users control over the automation by providing an expressive language and a convenient, easy-to-use user interface to specify the policies. The interesting and challenging aspect of the problem arises from the context in which the system operates. The end users are contract managers and buyers who are not trained in computers or programming. They enter their new supply contracts and define policy rules to control selection of the best contracts for buying requested parts. They act as their own knowledge engineers, even though the system is expected to have hundreds of rules for hundreds of contracts. The users interact with the system infrequently, perhaps only a few times a month when they begin or modify contracts, or change policies. Along with a moderate turnover rate of users, this makes it crucial that they can easily maintain correct rules with minimal training. In this paper, we describe a rule system and an interactive rule authoring tool designed to address the problems raised by this context. We believe these issues arise in most application domains where rule systems are put in the hands of the end users.

Introduction

The DEALMAKER sourcing module is designed to enable non-programmers to specify business policies that control automated supplier selection for off-the-shelf parts (those identifiable with part numbers).

User Organization

The Defense Logistics Agency (DLA) supplies US military forces and many federal organizations with the goods they need to carry out their operations in peace-time and during conflicts. It is responsible for over 3 million different kinds of consumable items and procures over \$15 billion each year in materiel. If it were commercial, it would rank in the top 75 of Fortune 500 companies. The customers of DLA submit requisitions for goods in electronic form. DLA arranges for the goods to be delivered from one of DLA's depots or directly from a commercial supplier. DLA bills its customers and pays the suppliers.

DLA is organized into Inventory Control Points (ICPs), which are broken down into Commodity Business Units (CBUs). A CBU manages a particular class of items, and has the flexibility to negotiate and manage contracts according to the needs of the particular industry segment. The people who carry out the business activities in a CBU are contract managers and buyers. Contract managers negotiate contracts with suppliers, and define the terms and restrictions for the contract (e.g., prices, delivery terms, shipping regions). Contract managers also define policies for managing the contracts (e.g., which class of DLA customers should use which contract). Buyers manage the daily operations of contracts, enter new contracts and revisions into the system, monitor the performance of vendors, and provide customer support.

A crucial aspect of the DLA business is the need to support a wide range of special requirements. These requirements arise from the nature of many of the items that they supply (e.g., hazardous materials, narcotic medicines, restricted technologies), special packing and shipping requirements, a complex set of priorities to support war-time and peace-time operations, and special needs of particular customers.

DLA's systems for electronic commerce do not provide the flexibility needed to address the wide range of special requirements and do not provide the agility to keep pace with the commercial developments in electronic commerce. For example, the special rules and policies for selecting among competing sources of supply are embedded in Cobol programs. Bringing up a new contract requiring special processing can take several months while their Information Processing department upgrades the relevant Cobol programs.

Challenging System Requirements

While the sourcing module is intended for continuous, automated use, changes to its database of supplier contracts and policies are entered manually. Some of the contract may be entered as rules because of the infeasibility of anticipating all of the complexity of the domain and representing it in conventional database tables. The contract-specific rules represent critical obligations and policies that must be implemented before the contract can be used for purchases. The need to enter

contract data and rules quickly suggests that programmer involvement needs to be eliminated. This leaves the end users, contract managers and buyers, to enter these rules. System administrators in each organization need to be able to engineer new rule types as well -- this is future work. Several design concerns arise from this:

- end users are not familiar with programming concepts;
- there are a large number of rules and contracts; and
- end users manage rules infrequently because contracts and policies evolve over long periods of time.

Contract managers and buyers are not expected to have training as a programmer. We have verified that this user population has difficulty with complex logical expressions, such as nested “ors” and “nots”. Such conditions are common on production rules, so the design must assure the users’ ability to manage rules in this system. Contract managers and buyers should not have to understand elements of the system out of their control such as the hundreds of rules on other contracts.

Contract managers and buyers each infrequently interact with the system to enter or modify supply contracts. They need a simple model for understanding function of rules as well as an intuitive interface.

The goal of the DEALMAKER system is to maximize direct control by contract managers and buyers over the policies and special requirements for managing contracts. The major challenge in the design is preventing the time spent learning to enter rules from being inappropriately large in proportion to the time spent entering rules. DEALMAKER takes the approach of providing a tiered framework that:

- maximizes the expressive capabilities and interactive support offered contract managers and buyers to enter their own rules without overtaxing their limited skills;
- provides for extension by non-programmer system administrators at the next level; and
- minimizes requirements for intervention by skilled programmers.

The following sections describe our design to meet this challenge, discuss the approach and contrast it with traditional AI systems, and state the current status of testing. One of the striking observations is how much can be accomplished with a simple paradigm at the end user and system administrator levels.

DEALMAKER: Designed to Meet the Challenge

From the requisition, the DEALMAKER sourcing module gets the part number, desired delivery date, priority code, customer, ship-to address, and a small set of attributes, such as “partial shipments not allowed” and “do not backorder.” It selects a supplier and provides all the data needed for an EDI Purchase Order and billing. The sourcing module has a database of supply contracts and

policies and has real-time access to some suppliers’ price and availability data. The sourcing module rejects requisitions it cannot match to a supplier due to lack of availability or contractual obligations.

To meet the challenging requirement of enabling non-programmer users to manage policies, we designed DEALMAKER with the following:

- a flexible contract representation;
- an easy-to-use interactive interface for contracts and their policy rules for use by contract managers and buyers;
- a template-driven interactive interface for rules covering organizational defaults and constraints for use by system administrators;
- a processing model simple enough for these users to understand their rules.

Planned, but not implemented, is the ability for system administrators to define new rule templates for use by themselves and end users.

Representing Contract Data and Policies

A contract is represented as a tree in XML that organizes attribute name-and-value pairs. At the root are values that are invariant for all parts supplied by the contract, such as the vendor name and address. Some nodes in the tree delineate groups of attribute values that serve as defaults when any of those attributes are not encountered at more detailed lower levels. For example, most, but not all, items are small and light enough to ship by express delivery. That will be the default in the parent group while just heavy and large parts specify otherwise at the item detail level.

Some nodes in the tree identify selector attributes. Their values at runtime select which branch below is to be used to look up additional attribute values. For example, when DLA is buying to replenish stock in their warehouses, the packaging and labeling requirements are for support of long-term storage with portions being distributed at different times. Purchases for direct delivery from the vendor to the customer and purchases for restock, therefore, have many different attribute values below a node controlled by the “method-of-support” variable. A Web browser-based contract editor has been developed enabling contract managers and buyers to enter and modify contracts using this structure.

A major goal met by this representation design is to minimize the redundant entry of contract data. The attribute values on one branch below a selector node are used to initialize those attributes on sibling branches. Only differences need to be entered. This is effective only when the display templates have been appropriately constructed.

New contract attributes can be defined in the XML and made available in the interface and to the rules without programming. However, considerable expertise is required to modify the contract DTD (XML template), the

display templates, and the XML file that maps attribute names used in rule conditions down through the selector nodes to the desired value.

Flexibility in specifying when a supplier is to be selected for various kinds of requisitions gives the contract managers leverage in negotiating better prices from suppliers. For example, promising an annual minimum purchase dollar amount allows the supplier to be sure of the ability to amortize overhead and, therefore, reduce unit prices. Similarly, a promise of all orders for the parts to maintain Caterpillar tractors in the Northeast may be an incentive for lower prices. It would be impossible to anticipate all these kinds of agreements, but it is possible to provide a rule mechanism allowing their implementation. The rule conditions match against attributes of the requisition and control which contracts may be selected. The rule action may prevent a contract from being used, indicate preference for a contract, or indicate exclusivity for a contract. These rules need to be entered and maintained by the contract managers and buyers in order to speed implementation of new contracts and to avoid costly and delaying involvement of programmers.

DEALMAKER has an interactive contract editor with highly tuned wizard-style panels designed for each specific kind of rule condition to make it as easy to use as possible. Most contracting policies are very easy to specify. This interface is implemented using XML, XSL, and DHTML and is embedded in the contract editor.

DLA is organized by commodity. Each DLA ICP manages a major category of parts, such as heavy equipment or construction supplies, and CBUs specialize further to commodity type, such as pharmaceuticals or hydraulic fittings. This organization implies specialized local defaults and constraints at the ICP and CBU levels, such as when to use the standard unit price and when to compute the price as DLA's cost recovery factor applied to the cost. While there are few ICPs, there are enough CBUs to require non-programmer system administrators be able to enter and maintain their local defaults and constraints with little training.

DEALMAKER uses Adaptive Forms (Szekely and Frank 1998), a grammar-based user interface that allows users to enter rules using structured English phrases. It shows users a form to fill in to specify an English paraphrase of a rule as seen in Figure 1. The system dynamically computes the fields in the form to include the fields that are compatible with the sentence fragment that the user has entered so far. This interface is implemented in Java and uses designs from form-based interfaces and NLMenu (Tennant, et al. 1983). NLMenu is a technique developed at TI in the early '80s which uses transition network grammars to dynamically generate a series of menus in such a way that users can enter only grammatically correct commands and queries.

The sourcing module uses the user-defined contracts, policies, defaults, and constraints in selecting a supplier for, or rejecting, each requisition.

The Phased Processing Model

DEALMAKER requires a simple processing model to allow non-programmers to specify rules. These users, contract managers, buyers, and system administrators, need to understand the functioning of their rules. We achieved this goal by partitioning the processing and the data in domain-specific ways. The processing is split into four phases, with phase one generating segments of data, phases two and three operating on these distinct segments of the data in succession, and phase four combining the results. The phases are introduced here and then more completely described.

1. *Generate proposals.* Create a proposal for each source of supply.
2. *Elaborate proposals.* Annotate each proposal with the attributes needed to compose a purchase order and bill the customer.
3. *Filter proposals.* Verify that each proposal meets the obligations and policies for use of its contract for the requisition. An annotation is added when a proposal needs special attention during ranking, such as exclusive rights for the sale.
4. *Rank proposals.* Sort proposals according to rule-selected criteria, such as fastest when the need is urgent or cheapest when the priority is low.

All processing is under the control of rules, except the generation of proposals. Elaboration and filtering rules focus on individual proposals. All rules whose condition is satisfied add an annotation to the proposal. The annotations for elaboration rules record computed attributes. The annotations for filtering rules record the effect of the filter, one of "must-use", "kill", or "null" (no effect). The annotations for the ranking rules record the criteria to use later in the ranking phase. The system applies the rules in the order determined by the data-flow dependencies among rules. This assures that any rule that tests a computed attribute is attempted after the rule that computes the attribute.

Generate proposals. Using the part number from the requisition as an index, the system queries its contracts database and the suppliers' online catalogs. The system constructs a proposal object representing each contract that supplies the item (and each alternative use of the contract, such as assuming a larger quantity to attain a better unit price). The initial attributes on the proposal come from the requisition, the standard data about the part, and the matched contract. The proposals are then considered individually in the elaboration and filtering phases.

Elaborate proposals. Elaboration rules implement organizational defaults and constraints. They compute and validate all attributes needed to compose the EDI Purchase Order transaction, an ANSI X.12 850 (ANSI X.12 Standard), and perform billing. These attributes include the cost to DLA, the marked-up price to DLA's

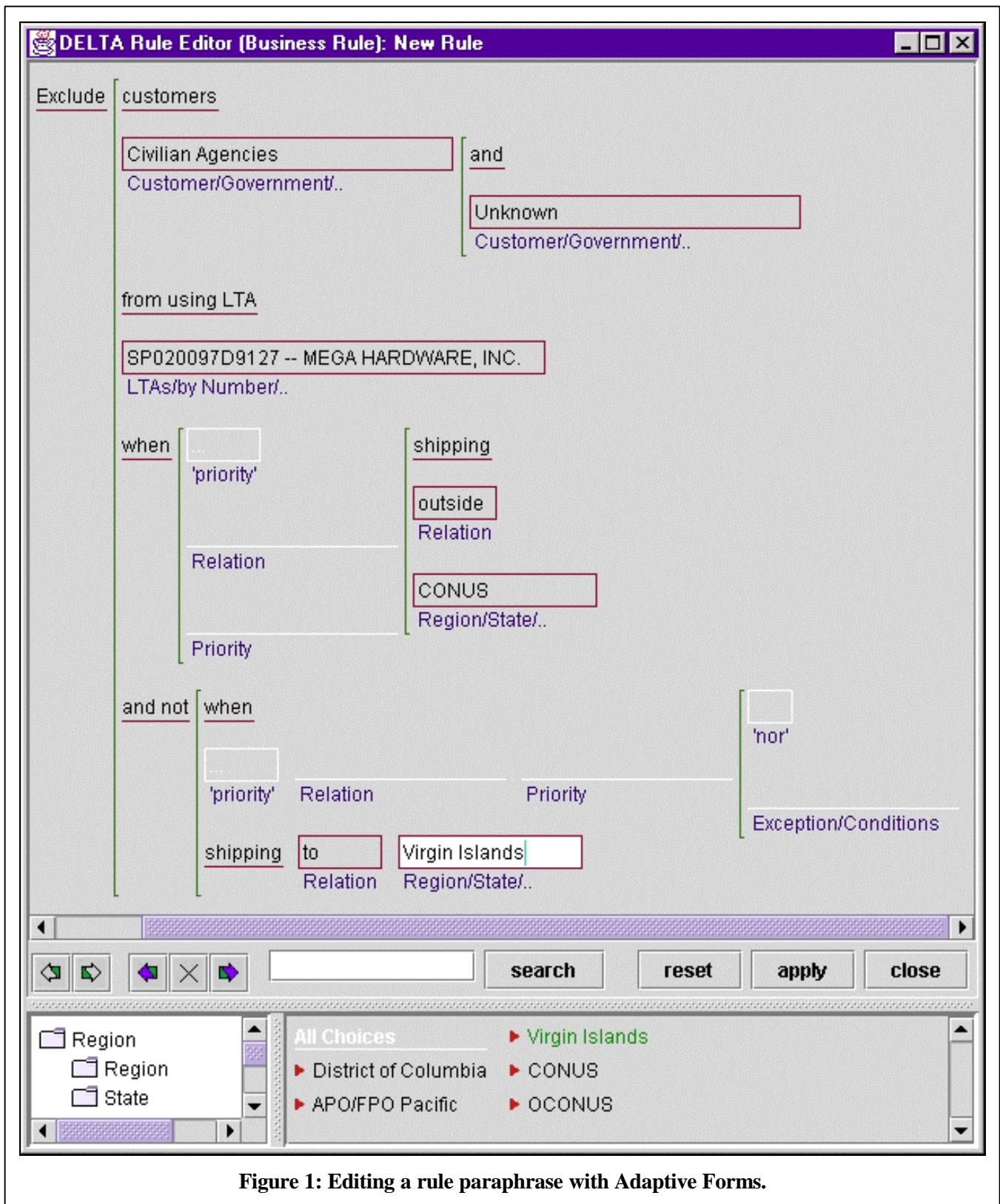


Figure 1: Editing a rule paraphrase with Adaptive Forms.

customer, and the expected delivery date. These rules process the requisition attributes such as “ship exact

quantity.” They implement policies and constraints such as allowing up to \$25 over standard prices and checking

customers' authorization to buy hazardous materials or pharmaceuticals. The maintenance of organizational default and constraint rules is intended for a system administrator and involves the customization of about 30 rules. These rules include some complex behavior.

One example is the computation of the quantity that must be ordered. This computation is complex because the units used in the requisition may differ from the units in which a vendor supplies the item, e.g., "ounce" in the requisition and "dozen" in the contract. Standard part data includes a unit of issue and the contract can record a conversion factor. In addition, the supplier might package items in quantities that do not match the requisition, such as a box of ten dozen. In order to align the units and deliver at least the quantity requested, rules make conversions for the purchase order, and they track any increase in cost to be sure it does not exceed the limit.

Another example is checking whether a quantity is close enough to the next price tier that it would be cheaper to order more. Our design makes this particular situation easy to handle. It creates an alternative proposal for each price tier in the contract. After the rules make any conversion needed to match units, another rule adjusts the quantity for the proposal's assigned price tier, killing those that cannot be simply increased to be within range. Because this changes the price, the rule checking the limit on excess will run after this rule. Ultimately, ranking will select the best of the surviving proposals.

The complexity of the elaboration phase means that system administrators need some training to manage the customization of defaults and constraints for their organization. Fortunately, these do not change often and they start with the rules for their parent organization. System designers construct the original set. An open issue is how the CBU rules are maintained if the original rules or ICP-customized rules they started with are modified. Fortunately, the rules specific to contracts do not overlap the functionality of these rules and are processed during the next phase.

Filter proposals. The filtering phase applies rules associated with a contract to verify that a proposal meets all that contract's policies. These rules annotate the proposal to kill it or grant exclusive rights. The ranking criteria are expressive enough to avoid the complication of granting preference.

Contract managers and buyers enter filtering rules, giving them the ability to express the obligations of their contracts and policies to control the source selection process. Three filtering rule templates have been defined.

- *Exclude.* These rules specify the conditions under which a requisition should be excluded from a contract or from specific items in a contract. For example, "Exclude all low priority, Army requisitions from contract X, but allow all requisitions from Fort Hood." Exclude rules place a "kill" annotation on the proposal.
- *Serves-only.* These rules specify that a contract only accepts requisitions with certain characteristics. For

example, "Contract X serves only low priority Army requisitions." Serves-only rules place a "kill" annotation on the proposal when the user-specified conditions are not met.

- *Must-use.* These rules specify that certain requisitions must use a specific contract or set of contracts. For example, "All low priority Army requisitions (for items supplied by contract X) must use contract X." Must-use rules place a "must-use" annotation on the proposal, which then receives special treatment during ranking in phase four.

DEALMAKER provides an easy-to-use interface allowing contract managers and buyers to specify filtering rules for their contracts.

Rank proposals. This phase starts when all proposals have either been through elaboration and filtering, or been killed. If all proposals have been killed, the requisition is rejected, otherwise, the live proposals are ordered best to worst. The same ranking criteria must be selected on all live proposals -- this is assured if the selection rules only check requisition attributes, such as priority code and desired delivery date. Proposal specifics, such as cost and days before shipping, are used during ranking. The ranking criteria may either sort the proposals by a continuous attribute, such as cost, or partition them according to a logical expression, such as "supplier can deliver within 30 days of the desired delivery date". Each criterion may specify next-level criteria for proposals equivalently ranked when it is done. A logical criterion specifies whether true or false identifies better proposals and can have different next-level criteria for true and for false. For example, the first criteria will usually partition any proposal annotated with "must-use" (hopefully by itself), and specify the criteria for the others when no "must-use" proposal is found..

Ultimately, either just one proposal is ranked highest, or the criteria run out leaving a set of equivalently ranked proposals. If used in an interactive sourcing application, several of the most highly ranked sources of supply can be presented to the user together with information that allows the user to make a selection, such as vendor, price, and days before shipping. For DLA's automated purchasing, equivalent proposals require a scheme to spread orders out fairly over time; we have not addressed this. The automated system is required to be fairly quick at selecting a source for each proposal and so our design was done with several optimizations in mind.

Optimization without Complication

The automated source selection module is required to keep up with the arrival rate of requisitions. DLA receives enough requisitions some days that they need to be processed at a peak rate of nearly 2000 per hour, or two seconds each at the bottleneck. Another requirement we adopted is preserving rule authors' simple processing model where all potentially relevant attributes appear on each proposal and all rules are attempted on every

proposal. These requirements taken together indicate the necessity for a technical solution to speed up processing. Traditional approaches, such as Rete nets, were rejected because of the large number rules and attributes, the sparseness of attributes on rule conditions, and the large range of values for some attributes. Retrieving or recreating the net at system initialization would be prohibitively time consuming. Three simple techniques make sense for this domain which, in combination, seem to have provided adequate speed (though this has not been measured).

First, rules are applied to one proposal at a time. This prevents rules from suffering complexities having to do with inter-proposal comparisons – all such comparison is left for the ranking phase where the rule-selected ranking criteria is algorithmically applied to the proposals. This also enables any proposal to be removed from consideration as soon as it receives a kill annotation.

Second, the attributes for a newly generated proposal are not actually copied to the proposal. Rather, all proposals share one copy of the requisition and standard part attributes. The contract attributes are accessed in the XML and then cached on the proposal when they are first referenced by a rule. This reduces the total amount of digging through the contract XML compared to prefetching all contract attributes that rules might reference. In addition, this on-demand lookup allows values for selection node variables to be determined by earlier rules. Note that because a proposal is for one supplier and only one proposal is considered at a time, there is no ambiguity with attribute names for values in the contract. Attributes added to the proposal by elaboration rules are kept only in the cache. Many of the values needed for the Purchase Order EDI message and billing outputs will be readily available in the cache.

Third, after the tens of ICP/CBU rules kept in memory are attempted, only those rules encountered while accessing the relevant data for the proposal are tried. Here we give an example of a rule that might be found in each of a contract header XML, a contract part XML, or common data about a standardized part. A contract header rule may grant that supplier exclusive rights to sell artillery spare parts to Camp. A contract part rule can kill urgent requisitions for that part -- a buyer would add such a rule as a result of poor past performance by that supplier in rapidly delivering this particular part. A standardized part rule can be used to prevent a nuclear detonator from being sold as assistance to a foreign country.

By tracking which rules are encountered while accessing a proposal's relevant data, exactly the rules with a chance of firing will be loaded and attempted. This way, the time to load rules is incurred incrementally and most rules may never be loaded in a given run. Once rules are loaded, they are cached in memory on the assumption that a number of these rules are likely to be used again during the run.

This section has described the techniques used in the DEALMAKER system. The goal is to enable the entry of

newly signed supply contracts by end users and to enable the implementation of new types of contractual relationships with suppliers by system administrators. We have addressed the representation of contracts as structured data in XML and unstructured policies as rules. We have presented our simple model of processing describing the phases and the rule engine. Optimizations transparent to this model were stated. The following section discusses how our designs compare with other AI technologies in the context of the goals and requirements of this source selection system.

Discussion

There are two subsystems to be evaluated for novelty and contribution. There is the automated source selector that finds the best supplier for each requisition and has to perform with reasonable speed and with a sizable database of supply contracts. And, there is the rule-authoring tool – the interfaces that support the entry of policies, defaults, and constraints that supplement the structured contract data. A simple model of understanding rule functionality, critical to the success of the system, drove the designs of the two subsystems. Getting enough expressiveness and speed for source selection had to be balanced against the skill required for users to manage their rules. We discuss the technology adaptations we devised to meet this challenge.

DEALMAKER: Automated Sourcing Module

Here, we evaluate our design for meeting the goals and requirements of the automated source selection problem, including scaling up to the real-world needs of DLA. We look at DEALMAKER as an AI application that combines adaptations of textbook AI techniques with the glue to make them flow together.

Solving the Real-world Problem. DEALMAKER was designed for production use by DLA, but has not been measured in that capacity yet. Care was taken to meet performance speed requirements using the optimizations described above. The XML contracts and rules are stored as long strings in an Oracle relational database with indexing on the part numbers. Timely availability and pricing data are obtained using a network connection to live data through PartNet (<http://www.partnet.com>), an aggregator of suppliers' catalogs. Sample elaboration rules, ranking criteria, and contracts with filtering rules have been entered to model the specific needs of the Defense Construction Supply Center, an ICP in Columbus, Ohio.

In our demonstration, the system appears to perform correctly and speedily when given test requisitions patterned after real ones. In addition, DEALMAKER has been demonstrated working as a plug-in module in a new system designed to receive DLA's electronic requisitions and connect with DLA's financial and EDI systems.

Adapting and Combining AI Techniques. Source selection can be viewed as a search for the best supplier from the universe defined in the database and the online catalogs. For this search, we selected the “Generate and Test” technique and added indexing by part number to guide the generation phase. “Generate” creates a proposal to represent each node in the search space. It includes elaboration, where each proposal is annotated with problem-specific data such as cost and availability. We divided “Test” into three separate phases: filtering performed during elaboration, filtering, and ranking. Filtering applies rules to each proposal in isolation. This avoids complex condition matching across multiple proposals.

Another AI technique is applied to make ranking fast, namely “Hill Climbing”, where an evaluation function applied to a state is used to move to a state closer to the solution. This might also be seen as “Best-first” or “Beam” search, where a search horizon moves toward the winning solution. When elaboration and filtering are complete, each live proposal has been annotated with delivery, cost, and other information important for evaluating the proposal’s quality for the requisition. Ranking applies the sequence of “evaluation functions” provided in the criteria to move closer to knowing which proposal is best with each step. AI has clearly contributed to the design of DEALMAKER. In return, DEALMAKER serves as a validation of a combination of adapted AI techniques in this problem domain. We feel a greater contribution is found in our designs supporting non-programmers as they manage rules to extend the functionality of the system.

DEALMAKER: Rule Authoring Tool

The design goal is to enable contractual relationships that reduce the government’s cost for parts. Support for current kinds of contracts and flexibility to define new kinds of contractual relationships between DLA and its suppliers are both crucial for the success of our system. Rules are used to represent contractual obligations and policies that are not easily represented as database fields. It must be easy for users to define rules to implement a variety of contracts and policies, such as preferred vendor. Ultimately, it must also be reasonably quick and easy to engineer new rule templates that are easy to use. In AI terms, users need a tool to move through the space of rules and rule templates to improve and extend the functionality of the automated sourcing module.

Our approach recognizes three tiers of knowledge-entering users: contract managers and buyers, also called end users; system administrators at each organizational level; and programmers. Contract managers and buyers enter contract-specific rules using a hand crafted, wizard-style interface. System administrators, with a bit more training, specify rules codifying defaults and constraints peculiar to their organization using the forms-based interface generated from rule template grammars. Global rules that implement security constraints and common

functionality are to be entered and maintained by a highly trained DLA system administrator with the assistance of a programmer. These rules may use templates and the grammar-driven editor, or may be coded for efficiency. Acceptance by the users will depend upon confidence that they are correctly controlling the automated sourcing process, ease of use, and understandability after going a long time since they last worked a contract.

We look at how our design was influenced by each of these concerns.

End users are not familiar with programming concepts. Users of DEALMAKER do not need to think about iteration or looping constructs. Their rules are applied to a single proposal to prepare it for ranking. The interface for contract managers and buyers provides templates that allow them to easily express the conditions that occur most often in practice rather than supporting full Boolean expressions.

Rules that require a broader view than just the contracts for a single commodity class or require deeper understanding of the sourcing process are reserved for system administrators. They manage these using the grammar-driven rule editor. We are hopeful that some system administrators will be able to work with complex Boolean expressions. However, while we have a grammar for arbitrary expressions of proposal attributes, we expect most administrators to rely on the specialized rule templates.

We have plans for a rule template editor for non-programmers to define new rule types for system administrators. Another capability we envision is an interface to create new wizard panels based on rule templates. This interface would allow system administrators to identify the rule conditions that need to be filled in by the wizard. To create rules based on that template, contract managers and buyers could run the wizard and provide the values needed to complete the rule conditions.

There are a large number of rules and contracts. Contracts and their rules are stored as individual entities in a database to permit the system to scale up to DLA’s real world requirements. To prevent rules from being a speed bottleneck, we took advantage of the domain-specific ability to partition the problem into proposals. Rules for other contracts will never apply to a proposal, so we index the rules in the database accordingly. Other rule indexes include the customer and shipping region. These indices prevent unneeded rules from slowing down a run of the sourcing module. The approximately 30 common rules that do not test an indexed attribute are kept in memory. This scheme works because rules do not interact across proposals and can, therefore, be retrieved individually. In practice, only a few specialized rules apply to each proposal. This design is for efficiency only – the contract managers and buyers may correctly have the mental model that all rules are tested for every proposal. The rules our system ignores would fail the condition by which they are indexed.

End users manage rules infrequently because contracts and policies evolve over long periods of time. The typical lifetime of a contract is five years. Revisions to the contract are expected every year, and occasional revisions may happen at arbitrary times. ICP-wide policies are expected to change seldom. Contract-specific policies may need to be established or changed quickly to respond to the demands of military operations. Contract managers and buyers do not manage a huge number of contracts each. Much of their time is spent negotiating new contracts and tracking and tuning the performance of existing ones. This makes rule managing a small and infrequent task. Even if users were able to recall their data and their training, it would be unacceptable for the interface to be difficult to learn and use. It would be equally unacceptable if users had to remember a large number of potentially interacting rules or had to gain knowledge of the contracts and policies managed by others. Users may be called upon to modify contracts and policies that they did not establish because of the need to react quickly and because of vacations and moderate user turnover.

The user interface supports this requirement by grouping rules with contracts, so when users open a contract they can see all the rules that affect processing of requisitions against that contract. Because rules for different contracts do not interact, users can quickly understand the policies that govern a contract, and have confidence that any modifications they make will not have adverse effects on other contracts. Likewise, eliminating rules that span multiple proposals, we eliminated the need to understand rules created by other users.

To keep training to a minimum, contract managers and buyers are not expected to understand or know the details of the ICP-wide rules. Also, we do not require these users to even think about rule ordering issues and effects derived from rule chaining. Instead, we developed a simple virtual model of operation based on the proposals and the phases. Representatives of DLA seem to understand how the rules they enter through the wizard panels in the contract editor will work.

Conclusion

The interesting aspect of the work is that by taking a multi-tiered approach that carefully constrains the production system architecture exposed to users, contract managers, buyers, and system administrators, we have insulated them from many of the well-known difficulties involved in building large production systems. Nevertheless, the architecture provides the flexibility and expressivity that gives contract managers and buyers with direct responsibility the ability to maintain their own contract data and rules. A system administrator in each ICP and underlying CBU has the ability to maintain the defaults and constraints special to that organization. Only as a last resort will intervention by a programmer be necessary. The system minimizes the effort and delay in

deploying new contracts, even new types of contracts, and thereby maximizes competition or otherwise minimizes the cost to the government for the billions of dollars spent each year in acquiring parts.

Current Status and Open Issues

We have built a prototype of the system described in this paper. We used a layered implementation strategy so that the DLA-independent features of the system are implemented in a reusable package that can be transferred to different applications. This reusable package includes the object representations for requisitions and contracts, the rule representation, and the rule application engine. We have gone through several iterations on the user interface based on demonstrations of the system to DLA personnel.

Our experience with the prototype, although limited, is encouraging. We have demonstrated the system with three contracts and about 30 rules. The system features most of the ICP-wide rules and several user policy rules. Extended validation still remains to be performed, but we have verified that the system produces valid EDI transactions for transmission to vendors.

The functional requirement to identify and select between substitute parts has not been addressed. We hope that the rule authoring tool could be used to let users control the search for substitute parts, even if there are many special cases. We did not develop an authoring tool for rule actions. Incorporating an interpreted expression language would decrease the need for the programmer and the associated delay in fielding new functionality.

Acknowledgements

This research was supported by the Defense Advanced Research Projects Agency under Contract No. DABT63-96-0066. Views and conclusions contained in this report are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of DARPA, the U.S. Government, or any person or agency connected with them.

References

- Frank, M. and Szekely, P. 1998. Adaptive Forms: An interaction paradigm for entering structured data. In *Proceedings of the International Conference on Intelligent User Interfaces*, 153-160, San Francisco, California.
- Tennant, H. R.; Ross, K. M.; and Thompson, C. W. 1983. Usable natural language interfaces through menu-based natural language understanding. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, 154-160, Boston, Massachusetts.