

DEALMAKER: An Agent for Selecting Sources of Supply To Fill Orders

Pedro Szekely, Bob Neches, David Benjamin, Jinbo Chen and Craig Milo Rogers
USC/Information Sciences Institute
Marina del Rey, CA 90292
(szekely, neches, benjamin, jinbo, rogers)@isi.edu
(310) 822-1511

Abstract

The central problem that we seek to address is: How does the average person communicate complex intent to an agent? A number of factors make this difficult. (For a more detailed discussion of these issues, see Neches, 92.) These include: limited understanding of the agent's capabilities, limited understanding of the language available to instruct the agent, limited ability to think and communicate in terms of algorithms, limited ability to analyze interactions between high-level requests. Our approach attempts to offer a sensible alternative. It is built upon two key ideas: (1) the *Business Rule Manager*, a set of mechanisms for providing *constrained dialogues* between user and system; and, (2) and a *tiered interaction framework*, a system that supports *social evolution of communications* by helping a community of users help each other develop a common language for talking with their agents. We present our approach to this problem in the context of DEALMAKER, an agent for selecting the best source of supply to fill orders. It receives requisitions for items in electronic form, queries a database of preferred vendors to determine possible sources of supply, and applies rules to filter and rank these sources according to policies established by contract managers.

User Organization

The ideas discussed in this paper have been developed and tested in the context of applications for the Defense Logistics Agency (DLA). This organization 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.

A critical 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 the special needs of particular customers.

DLA's systems for electronic commerce do not provide the flexibility needed to easily address the full 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 until their Information Processing department upgrades the relevant Cobol programs.

Although there are fallbacks in the form of intervention by system administrators or programmers, 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. DEALMAKER offers the following capabilities:

- A flexible representation scheme for contracts based on XML.
- A sourcing module that interprets business rules to select an appropriate source of supply to fill a requisition.
- An interactive user interface to put contracts online and to enter the rules that govern the use of the contract.

The challenge is to provide a layered framework that: (1) maximizes the expressive capabilities offered buyers and contract managers to program their own rules without overtaxing their skills; (2) provides for next-level extension by system administrators rather than full programmers; and (3) minimizes requirements for intervention by skilled programmers. The following sections describe our approach, discuss the current status of testing, and evaluate the approach against the difficulties posed for traditional AI systems. One of the striking observations is how much can be accomplished with a simple paradigm at the user and system administrator levels.

The DEALMAKER Sourcing Agent

The DEALMAKER sourcing agent is part of a larger society of agents that automates the processing of requisitions for DLA. The sourcing agent receives requisitions from other agents and determines the best source of supply for each requisition. Other agents send the orders to the vendors using the appropriate EDI transactions, handle the financial transactions with the customers and the vendors, and handle vendor responses.

The sourcing agent is implemented internally as a blackboard / production architecture organized around these three tiers of developers. The core is an interpreter of grammar-based rule templates and instances, which may be extended by programmers to add new low-level functionality. Following the constraints of the grammar, system administrators can create templates that define new classes of rules. Following those templates, buyers and contract managers can create new instances of these classes. Details and examples follow.

The sourcing agent processes requisitions in four stages:

- 1) *Generate proposals.* Using the item identification from the requisition, the system queries the database of contracts and an external source of vendor catalogs, such as PartNet, to find all contracts that supply the item. For each match, the system constructs a proposal object that represents the possibility of using that contract to supply the item. All attributes of the requisition and relevant attributes of the matched contract are copied onto each proposal.
- 2) *Elaborate proposals.* Compute and validate all attributes needed to compose the EDI transaction to submit an order to a vendor, including cost to DLA, marked-up price to DLA's customer, and expected delivery date. Rules implement DLA standard policy and contract-specific exceptions.
- 3) *Filter proposals.* Apply rules to verify that a proposal meets all the particular policies that have been established to use a particular contract for the particular requisition that needs to be filled.

Filter rules annotate the proposal with recommendations to kill it or to use it in preference of other proposals.

- 4) *Rank proposals*. Apply rules that organize proposals into equivalence classes based on the requisition's priority code and desired delivery date. The rules successively split equivalence classes into smaller classes. The winning proposals are found by traversing the resulting tree of equivalence classes to find the first non-empty leaf class.

The processing for all steps except the generation of proposals is under the control of rules. The elaboration and filtering rules work on individual proposals. The system processes one proposal at a time. The system applies the rules according to data-flow dependencies among rules. This assures that rules that test a computed attribute are attempted after the rule that computes the attribute. All rules whose condition is satisfied are applied, adding an annotation to the proposal. The annotations for the elaboration rules record computed attributes, the annotation for the filtering rules record the effect of the filter (e.g., kill) and the annotation for the sorting rules record the equivalence class to which the proposal is assigned.

In the DLA application, the elaboration, ranking and most filtering rules are defined by each department, called an Inventory Control Point (ICP), and are expected to be handled by system administrators rather than by contract managers or buyers.

Specifying Policies To Agents

The user interface for entering policy rules allows users to enter rules using structured English phrases. This interface uses elements from form-based interfaces and NLMenu¹, to show users a form that people can fill in to specify an English paraphrase of a rule [Frank 98]. 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.

Our interface paradigm is well suited for instructing agents because it combines many of the strengths of natural language and form-based interfaces, while overcoming their main limitations. Requests in the structured English approach look very much like normal English sentences, making it easy for users to formulate their requests, and to understand requests that have been entered in the system by others. Our approach also avoids two of the main problems with natural language interfaces. First, there is no recognition problem or need for clarification dialogues. The menus show users the words that the system can understand, and the form fields show users the structures that are valid. Second, the interface prevents users from formulating requests that the agent cannot perform. Most agents have limited functionality, and it is always hard for users to understand the boundaries of the agent's expertise. Using natural language, it is very easy for users to formulate requests in the general application domain, but which the agent cannot carry out. Our forms prevent users from specifying such requests. The form and field prompts show users the structure of the requests that the system can understand. The fields and prompts remind users to enter information that the system needs to carry out a particular kind of request, and also remind users about the limitations of the agent – if there is no field for something, the agent doesn't understand it.

¹ NLMenu is a technique developed by Craig Thompson 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 only generate grammatically correct commands and queries.

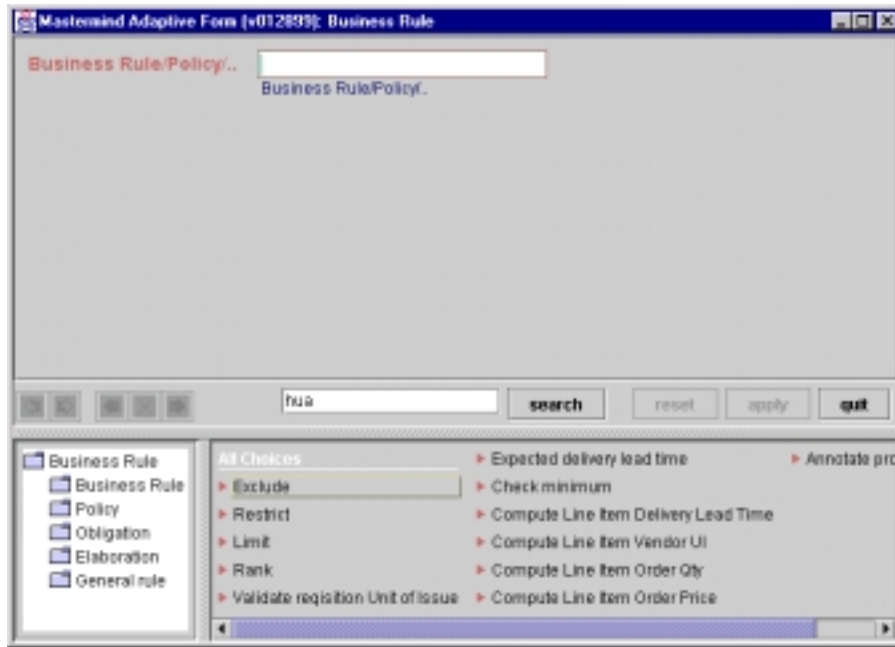


Figure 1. Rule editor screen for selecting the type of business rule to create.

The screen in Figure 1 shows the form for entering rules. The user can select the type of rule by selecting from the menu at the bottom of the screen, or by typing in the box at the top of the screen. Suppose the user selects the word 'Exclude', to enter an exclusion rule.

Figure 2 shows the screen after the user specifies that a customer should be excluded. The system

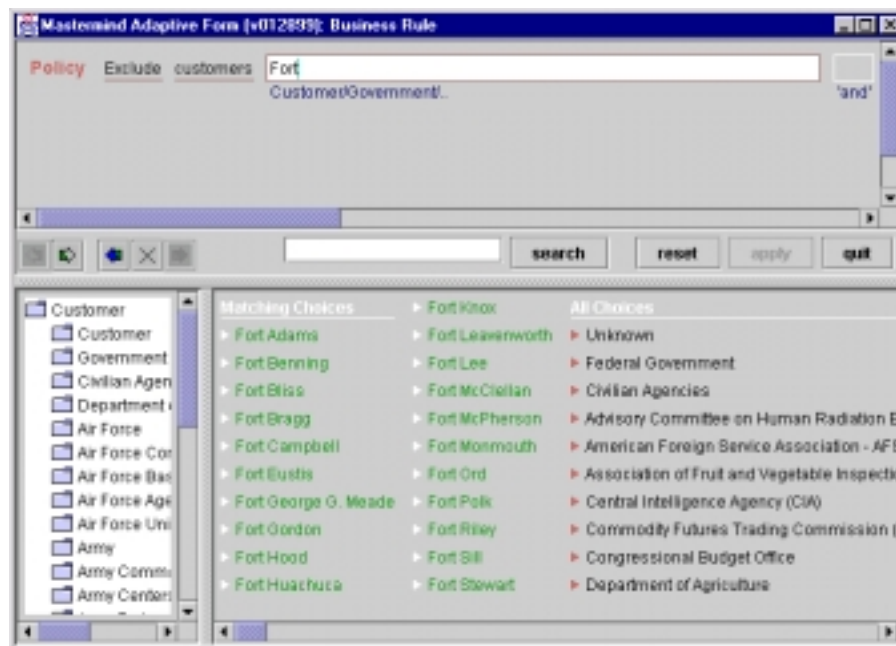


Figure 2. The user has elected to create a customer exclusion rule. The user is now specifying the customer or group of customers that should be excluded.

now shows the user the fields for entering the information required for specifying a customer exclusion rule. The user has typed the word 'Fort', and the bottom panel shows all the matching customers. The user can select a choice with the mouse or type additional characters until the choice is unique.

Figure 3 shows the form after the user has specified the customer, the contract from which the customer should be excluded, and the priorities when the order should be excluded. The user is now working on specifying a condition based on the shipping location for the order.

Once the user finishes entering the region for the shipping location clause, the rule will be complete, the 'Apply' button will be enabled, and the user will be able to install the rule in the database.

The range of rules and conditions that users can enter is specified using a grammar in a notation similar to BNF. The system tracks the user inputs against the grammar and computes all the possible continuations for the sentence fragment that the user has entered so far. Based on the continuations, the system constructs fields to let the user select how to complete the sentence, and shows a menu of the possible fillers for the current field. When several continuations are possible, the system lets the users enter them in any order. For example, the priority and shipping conditions are independent of each other, so the user can enter them in any order.

The system provides forms similar to those shown in this example to enter the three main filtering rules that contract managers and buyers use to manage contracts (exclusion, must-use and serves-only rule types), and a few of the ICP-wide elaboration, filtering and ranking criteria.

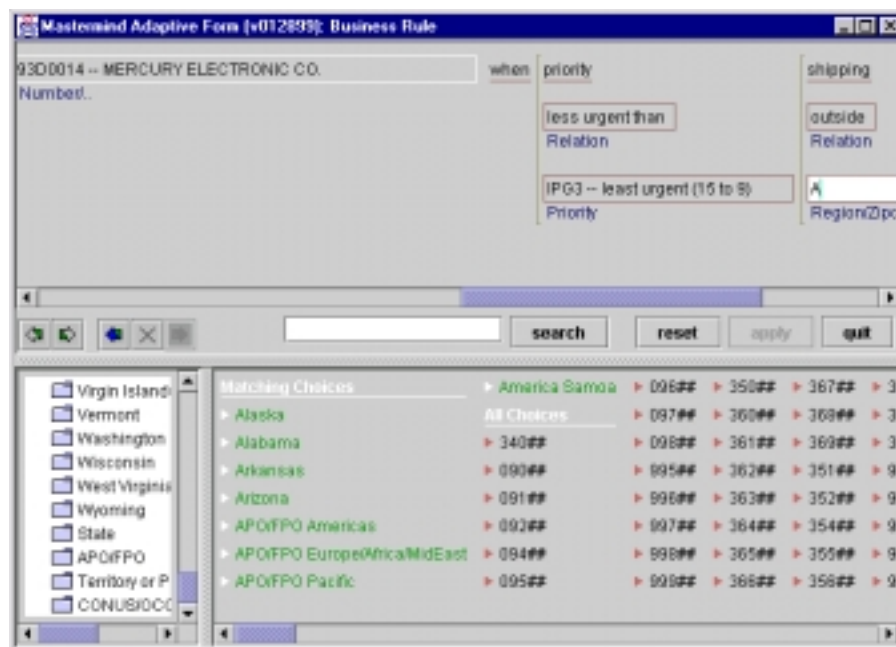


Figure 3. The rule editor screen after several steps where the user selected the contract from which the customer should be excluded, and specified the requisition priorities when the exclusion should take place. The user is now specifying a condition based on where the goods are to be shipped.

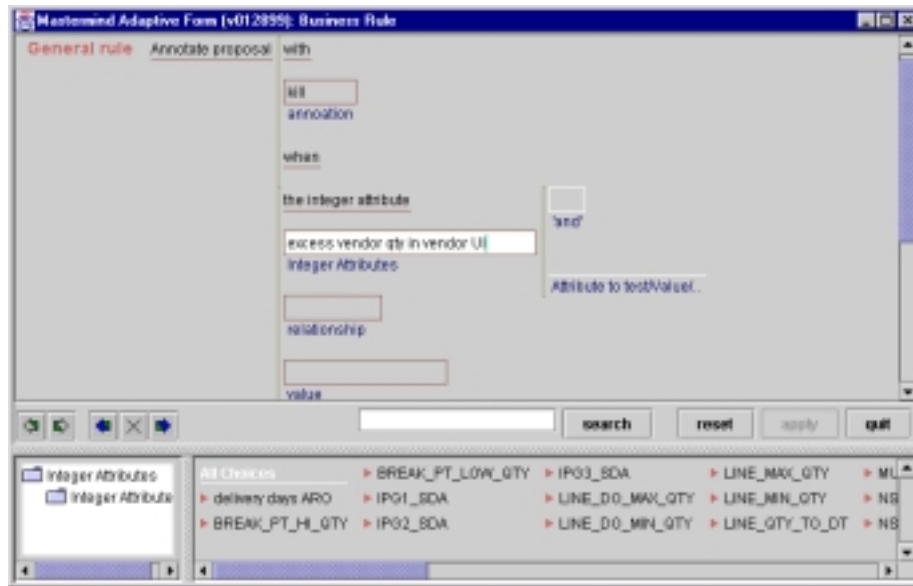


Figure 4. Screen for system administrators that enables them to specify a wider range of rules, by directly defining the annotation to place on a proposal and by referring to the proposal attributes to define the conditions for rule application.

The system also allows more advanced users to enter a wider range of rules by using the 'Annotate proposal' rule type (Figure 4). The interaction paradigm is the same, but the grammar is at a level of abstraction below the other types. The grammar and vocabulary for this rule allows the user to express the rules by directly specifying the annotation to place on the proposal, and the attributes and conditions to be tested. This interface is intended for system administrators who may be expected to understand the meaning of the attributes. The screen in Figure 4 shows an example of a user entering a rule that kills a proposal when the value of the attribute 'excess vendor qty in vendor UI' is in some relationship (e.g., exceeds) to another value.

DEALMAKER offers the capability to extend the grammar for expressing rules using the end-user terminology. The process involves two steps. The first step, which can be accomplished by a system administrator, is to extend the grammar by adding new structure and terminology. The second step is to define a template that maps the new terminology into the relevant proposal attributes and Boolean expressions that define a rule. Currently, this step involves defining a new Java class, and hence must be performed by a programmer. We are working towards a template editor that would allow a system administrator to define the mapping using an interactive user interface.

This capability supports a kind of social evolution of communication that our scheme is intended to foster. It's very difficult and unrewarding -- and therefore very unattractive -- for most users to spend time figuring out how to instruct their agents. It's also very difficult for developers to successfully provide users with the "right" language for doing so (particularly given the need to balance between expressive power and convenient simplicity). Our alternative is to simply get in the ballpark, and let the community help each other to work out the details.

Discussion

One of the most interesting aspects of our work are the tradeoffs we made between expressiveness and ease-of-use of the rule engine and rule user interface. A similar analysis of these tradeoffs was

published by Grosz [Grosz 96]. While we believe that through careful design it is possible to construct systems that score well on both dimensions, the tradeoffs are unavoidable. Our goal was to build a system that provides access at multiple entry points along the expressiveness/ease-of-use continuum. We offer two related user interfaces. One, directed at contract managers and buyers, is easier to use but less expressive. The other is harder to use because it requires a better understanding of how information is encoded in the system. However, it offers significantly more expressive power, and is intended for system administrators. The interaction paradigm in both interfaces follows our form-based, structured natural language approach.

The tradeoffs between expressiveness and ease of use are reflected not only in the user interface, but also in the design of the rule execution engine. We believe it is not possible to mask system complexity through clever interface design. The execution model of our rule engine is very simple:

- Rules only refer to a single proposal. Users can look at the rules defined for a contract and be confident that they do not interact with the rules defined for other contracts.
- All rules that apply to a proposal are executed once. The only looping construct of the system is the step that generates proposals. Users need not worry about controlling rules to apply to a whole collection of objects or to avoid them firing more than once.
- The system determines the order for applying rules based on data-flow requirements. The system orders rules so that rules that produce the value of an attribute are executed before rules that test the attribute in their conditions. Loops are errors. Users need not worry about the order in which rules are executed.

In summary, our approach for defining rules addresses the four factors that make it difficult for people to instruct agents:

Limited understanding of the agent capabilities. The user interface forms allow users to visually explore the space of requests that the agent can understand. When users select a type of request, the form updates to show the fields relevant to the particular type of request. When the user selects a field for a complex parameter, it will also expand to show the different options that can be set.

Limited understanding of the language to instruct the agent. Users do not need to learn a specialized language for entering requests to the agent. The interface allows users to enter requests in restricted natural language. The forms show the users the words and terms they can use to construct sentences, and guide users to construct correct and complete requests.

Limited ability to think and communicate in terms of algorithms. The interface for contract managers and buyers allows these users to specify their requests as simple sentences using terms familiar to them. Users are not required to understand algorithmic constructs such as loops and complex Boolean expressions. System administrator can state more complex rules in terms of attributes of proposals, comparison operators and complex Boolean expressions.

Limited ability to analyze interactions between high-level requests. The rule language is design to avoid interactions between rules. Each rule affects only a single proposal, and users do not need to worry about the order in which rules are applied.

It is important to note that this paper illustrates our interface paradigm in the context of an electronic commerce application. The rule engine, and the interface technology are generic and could be used to construct agents in other domains.

Current Status

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.

Our experience with the prototype, although limited, is encouraging. Our system can represent real contracts, and our rule specification language and grammar support all the elaboration and filtering rules needed to produce valid EDI transactions for vendors, and a wide variety of contract manager and buyer defined policies. These include the three basic relationships between requisitions and contracts (exclusion, must-use and serves-only) and conditions referring to customer, ship-to location, priority, project, and a wide range of flags.

We have demonstrated the system with three contracts and about 30 rules instances representative of rules that users may be interested in defining for the three sample contracts. It is worth noting that the relatively small number of rules is not a reflection on the small size of the effort, but rather an illustration of the simplifications and improvements possible in a framework that supports social evolution of communications. The application initially had over a dozen rule classes and nearly a hundred rules, but restructuring of the classes drastically reduced the number of types and instances that had to be exposed to the non-programmer tier of users. A system with more contracts would have correspondingly more rules, but the complexity is still significantly less.

We propose to demonstrate the system during the workshop. We will bring a laptop that runs a stand-alone version of the system.

References

[Frank 98] Martin R. Frank and Pedro Szekely. **Adaptive Forms: An interaction paradigm for entering structured data.** In Proceedings of the ACM International Conference on Intelligent User Interfaces, pages 153-160, (San Francisco, California, January 6-9) 1998.
<http://www.isi.edu/~frank/Papers/iui98.pdf>

[Grosz 96] Benjamin N. Grosz. **Approaches to Authoring of Rules for Intelligent Agents** In Proceedings of the 1996 [AAAI](#) Spring Symposium on Acquisition, Learning, and Demonstration: Automating Tasks for Users, edited by Yolanda Gil. Held at Stanford University, Stanford, CA, USA, Mar. 1997. AAAI Technical Report SS-96-02, AAAI Press, Menlo Park, CA, USA.
<http://www.research.ibm.com/people/g/grosz/paps/aacqsp.pdf>

[Neches 92] R. Neches. *Cognitive Issues in the SHELTER Knowledge Base Development Environment.* **AAAI Symposium on Cognitive Issues in Knowledge Acquisition**, 1992.

