Explanation Structures in XSEL

Karen Kukich
Computer Science Department
Carnegie-Mellon University
Pittsburgh, PA 15213
412-578-2621
Kukich@CMU-CS-A

1. Introduction

Expert systems provide a rich testbed from which to develop and test techniques for natural language processing. These systems capture the knowledge needed to solve real-world problems in their respective domains, and that knowledge can and should be exploited for testing computational procedures for natural language processing. Parsing, semantic interpretation, dialog monitoring, discourse organization, and text generation are just a few of the language processing problems that might take advantage of the pre-structured semantic knowledge of an expert system. In particular, the need for explanation generation facilities for expert systems provides an opportunity to explore the relationships between the underlying knowledge structures needed for automated reasoning and those needed for natural language processing. One such exploration was the development of an explanation generator for XSEL, which is an expert system that helps a salesperson in producing a purchase order for a computer system [10]. This paper describes a technique called "link-dependent message generation" that forms the basis for explanation generation in XSEL.

1.1. Overview of XSEL

Briefly, the function of the XSEL system is to assist a salesperson in configuring a custom-tailored purchase order for a Digital Equipment Corporation VAX computer system. XSEL works with the salesperson to elicit the functional computing requirements of the individual customer, and then goes on to select the components that best fit those requirements. The output of an XSEL session is a purchase order consisting of a list of line-items that specify hardware and software components.

There are two main phases to XSEL's processing, a fact gathering phase and a component selection phase. During the fact gathering phase XSEL carries on an interactive dialog with the salesperson to elicit values for facts that determine the customer's functional computing requirements. These might include requirements for total disk space, percent of removable disk storage, number of terminals, lines-per-minute of printing, etc. Natural language processing during the fact gathering dialog is minimal; XSEL displays menues and pre-formulated queries and accepts one- or two-word answers from the user.

Once enough facts have been collected XSEL begins a silent phase of processing. During this phase a set of candidate components that satisfy the customer's basic requirements is retrieved from the DEC parts database. Within each class of component, i.e., processor, disk, terminal, etc., candidates are ranked according to their score on an evaluation function that measures the degree to which a candidate satisfies the customer's weighted functional requirements. The candidate with the highest score is selected and placed on the purchase order.

The most important knowledge structure used by XSEL during the fact gathering phase is a fact. A fact is simply a list of attribute-value pairs that represent knowledge about one of the customer's functional computing requirements. Figure 1-1 depicts a sample fact.

```
(FACT ↑ATTRIBUTE TOTAL-DISK-SPACE
      ↑STATUS INFERENCE ↑CLASS DISK
      ↑UNITS MEGABYTES ↑MEAN 3600
      ↑TOKEN G:29)
```

Figure 1-1: Sample XSEL Fact

The fact collection process is driven by backward-chaining rules. A top-level rule deposits a few "core" facts for which XSEL must obtain values, such as "total-disk-space", "total-number-of-terminals", etc. One at a time, XSEL solicits a value for these core facts from the salesperson. If the salesperson answers "unknown" to a solicitation, another rule fires to deposit some additional facts that would enable XSEL to infer a value for the unknown fact. The cycle is then repeated as XSEL solicits values for each of the newly deposited facts. Any time a newly instantiated fact completes the set of facts required to infer a value for some other fact, the appropriate inference rule is automatically triggered and the value for another fact is inferred. This backward-chaining process continues until XSEL obtains values for all of the core facts, or until no more data can be collected and no more inferences can be made, in which case some default value rules fire to instantiate values for any remaining unknown facts.

The most important knowledge structure used by XSEL during the component selection phase is a rank element. Like a fact, a rank element is simply a list of attribute-value pairs. In this case the attribute-value pairs represent knowledge about a candidate's score for one term in the evaluation function. A different evaluation function is associated with each class of component, and each evaluation function is a sum of some weighted terms. The terms of the evaluation function for the class disk, for example, include price, disk-pack-type, storage-capacity, average-access-time, peak-transfer-rate, and handedness. For every candidate, XSEL computes a rank value for each term in the evaluation function. The rank value for a term is the product of the candidate's normalized score for the term and a weight which represents an importance factor. The essential information needed to compute a rank value for a term for a candidate is stored in a rank element, an example of which is shown in Figure 1-2.

```
(RANK  ↑RANK-NAME AVERAGE-ACCESS-TIME
    ↑NAME RA60-AA* ↑CLASS DISK
    ↑RANK-VALUE -3 ↑COEFFICIENT 1
    ↑VALUE 50 ↑IMPORTANCE 1
    ↑TOKEN G:9)
```

Figure 1-2: Sample XSEL Rank

After all the rank values have been computed for a candidate they are summed to obtain a total score for the candidate. The candidate with the highest total score is selected and placed on the purchase order.

The component selection phase is driven by forward-chaining rules. These rules perform the subtasks of first, retrieving candidates from the database, next, determining a quantity and cost for each of the candidates, next, computing a total rank score for each candidate, and finally, selecting the candidate with the highest rank score.

At present, the entire XSEL system consists of over three thousand OPS5 [2] rules. The explanation generator, which will be described shortly, comprises an additional five hundred rules. Anywhere from approximately five hundred to five thousand rules may fire during the fact gathering phase to create from fifty to five hundred facts, and roughly three thousand rules will fire during the component selection phase to create around one thousand rank elements. The whole process can take anywhere from ten to thirty minutes of real time, depending on how XSEL's queries are answered.

1.2. Sample Explanations

Three of the most obvious types of queries a user might ask were targeted for initial explanation development. Sample explanations from each of those types are given in this section. The following sections describe the knowledge structures and processes within both XSEL and the explanation generator that produced those explanations, as well as the goals and rationale behind them.

One type of query that is likely to be asked is why a particular component appears on a purchase order. We refer to queries of this type as "why-choice" queries. To answer a why-choice query the explanation generator must compare the rank elements for each candidate on each term of the evaluation function in order to determine which attributes were responsible for the higher score of the component that was actually selected. The following are sample explanations from the why-choice class of queries.

? why ra81

THE RA81 IS CHEAPER THAN ANY
ALTERNATIVE FIXED PACK DISK ,
POSSIBLY BECAUSE IT HAS A SMALLER
TOTAL STORAGE CAPACITY AND A
SLOWER AVERAGE-ACCESS-TIME .

? why rm05

ALTHOUGH THERE ARE LESS EXPENSIVE
DISK S , THE RM05 HAS A LARGER
DISK PACK THAN ANY ALTERNATIVE
REMOVABLE PACK DISK .

Figure 1-3: Sample Why-Choice Explanations

A second obvious type of query asks why a certain fact has whatever value it has, e.g., why total-disk-space is 3600 megabytes. We refer to queries in this class as "why-fact" queries. In the case of why-fact queries, the explanation generator must examine the facts that were created during the fact gathering phase, and it must determine how those facts are related through the backward-chaining process. An example of an explanation that was generated in response to a why-fact query follows:

? why q total-disk-space

XSEL INFERRED A VALUE OF 3600 MEGABYTES
FOR TOTAL-DISK-SPACE . 3574 MEGABYTES
ARE REQUIRED FOR TOTAL-USER-DISK-SPACE .
THE REMAINDER IS ACCOUNTED FOR BY OTHER
FACTORS , SUCH AS SUM-OF-SYSTEM-DISK-
SPACE.

3574 MEGABYTES WAS INFERRED FOR
TOTAL-USER-DISK-SPACE BECAUSE 2859
MEGABYTES ARE REQUIRED FOR USER-DISK-
SPACE AND THAT VALUE IS MULTIPLIED
BY 125 FOR PERCENT-FOR-EXPANSION .

XSEL INFERRED A VALUE OF 25 MEGABYTES
FOR SUM-OF-SYSTEM-DISK-SPACE FROM 1
SYSTEM-DISK-SPACE REQUIREMENT OF 25
MEGABYTES FOR THE VMS OPERATING-SYSTEM .

Figure 1-4: Sample Why-Fact Explanation

This explanation would have ended immediately following the first paragraph had not the user previously asked for longer explanations. But because the user had earlier typed "explain more", the explanation generator went on to explain the terms "total-user-disk-space" and "sum-of-system-disk-space", which were introduced in the first paragraph. If the user were to type "explain more" a second time, and then ask the same question "why quantity total-disk-space", the explanation generator would not stop where it did. Instead, it would go on to explain the terms user-disk-space, percent-for-expansion, and system-disk-space, which were introduced in the second and third paragraphs.

There is no upper bound on the number of levels of explanation the user may request. If the number of levels to explain is high, XSEL will keep explaining until it reaches those facts whose values were set either by user input or by default, in which case there is nothing further to explain. The user can also type "explain less" at any time, thus decreasing the number of levels to explain. The lower bound on the number of levels to explain is one.

The mechanism for determining which term to explain next is a queue. As new terms are introduced they are placed in the queue. The queue was originally implemented as a stack, but as explanations got longer they began to sound less coherent using the stack mechanism. So the queue was implemented, but the stack was retained. Now one can toggle between them by typing "explain queue" or "explain stack", thus producing alternatively structured explanations for the sake of comparison.

The third obvious class of queries asks why a certain quantity is needed for any line-item. We refer to these as "why-line-item" queries. Why-line-item queries require the most complicated processing because the explanation generator must understand how the line-item that was selected relates back to the facts that determine the quantity needed, and there is usually a long sequence of forward-chaining rules as well as the whole evaluation function mechananism between the creation of the facts and the creation of the line-items. Figure 1-5 shows a sample explanation from the why-line-item class. In this example, the number of levels to explain was set at two. The first two paragraphs comprise the first level, so the explanation could have

stopped there; the remaining two paragraphs were generated in response to terms introduced in the first two paragraphs.

```
? why q ra60

4 RA60-AA* 'S WERE SELECTED  IN ORDER  TO
SATISFY A REMOVABLE-DISK-SPACE
REQUIREMENT OF 900 MEGABYTES .

EACH RA60-AA* PROVIDES A CAPACITY OF 205
MEGABYTES .  THEREFORE ,  4 RA60-AA* 'S ARE
REQUIRED TO YIELD AT LEAST 90 PERCENT OF
THE REMOVABLE-DISK-SPACE CAPACITY OF 900
MEGABYTES .


900 MEGABYTES OF THE TOTAL-DISK-SPACE
REQUIREMENT OF 3600 MEGABYTES WERE
ALLOCATED  TO REMOVABLE-DISK-SPACE .
XSEL INFERRED A VALUE OF  900 MEGABYTES
FOR REMOVABLE-DISK-SPACE  BECAUSE  3600
MEGABYTES ARE REQUIRED FOR TOTAL-DISK-
SPACE  AND  2700 FIXED-DISK ARE
SUBTRACTED FROM IT TO GET THE DIFFERENCE .

THE VALUE OF 205 MEGABYTES FOR REMOVABLE-
DISK-UNIT-CAPABILITY WAS RETRIEVED FROM
THE DATABASE .
```

Figure 1-5: Sample Why-Line-Item Explanation

2. XSEL Explanation Design Goals

2.1. Related Explanation Work

The design of the XSEL explanation generator was motivated by three goals: first, that explanations should be accurate, second, that explanations should be direct, and third, that some degree of generality should be attempted.

Most early attempts at explanation generation adopted either a canned text or an execution trace approach. The canned text approach led to accuracy problems and the execution trace approach led to directness problems. These problems are described in detail by Swartout [12]. In brief, canned explanations can suffer from a lack of accuracy in the event that any modifications or additions are made to the performance program without the corresponding modifications or additions being made to the canned text. Execution trace explanations tend to suffer from a lack of directness because every step during program execution gets reported, including what Swartout has referred to as "computer artifacts", as in "Variable X was initialized to 0".

Another common early approach to explanation generation was the goal tree approach, which is very similar to the execution trace approach. The original explanations produced by the MYCIN system were goal tree explanations [1]. This approach allowed the user to question any request for information made by the system, and the system would simply locate the goal immediately above the current one in the goal tree and report that it needed the information to resolve that higher goal. Goal tree explanations tend to suffer from the same lack of directness problems that execution trace explanations suffer from.

Swartout's work on an explanation generator for the Digitalis Therapy Advisor attacked the accuracy and directness problems successfully. His approach was to redesign the DTA, separating descriptive facts from domain principles and from the abstract goals of the system. This allowed the performance program to be generated by an automatic programmer, which also created a goal refinement structure in the process. The goal refinement structure captures the knowledge that goes into writing the performance program, and makes it accessible to the explanation generator, where it can be used to produce explanations that are both accurate and direct. Furthermore, as Swartout points out, such explanations can be viewed as "justifications" for the system's behavior.

One of the major contributions of the DTA work was to demonstrate that a single explicit representation of knowledge can and should drive both the automatic program generation process and the explanation generation process. Further research supporting the "shared explicit knowledge" approach to automatic knowledge acquisition, rule generation, and explanation generation is underway for at least three other projects [8] [4] [5] [6].

2.2. The XSEL Explanation Approach

XSEL's approach to explanation generation differs from all of

the approaches discussed above. The sheer size of XSEL would make implementing canned responses tedious. Similarly, the number of rule firings on any run would make reading execution trace explanations laborious even, or perhaps especially, if they were translated into natural lanaguage. The approach taken by Swartout of extracting the regularities and representing them separately as domain principles would work for the backward-chaining rules used during XSEL's fact gathering phase, but the forward-chaining rules used during the component selection phase are so irregular that attempting to extract regularities would result in the duplication of nearly the entire set of rules. Some other common denominator needed to be found in order to achieve some computational power for explanation generation. For about two thirds of XSEL's explanation facilities, that computational power was bought by the creation of links, which are simple knowledge structures that establish relations between elements in XSEL's working memory. The role of links will be the focus of the remainder of this paper. But first a brief general overview of all the explanation facilities is given.

There is a simple variant of a goal tree explanation facility built into XSEL, so that the system can always state why it wants a value for any fact it requests during the fact gathering dialog. But the explanation samples shown in the previous section were generated by an entirely different mechanism, a message-based explanation generator. A message-based explanation generator is a two-phase processor that first generates and organizes messages based on the contents of working memory, and then maps those messages into surface strings. Two different types of message generator have been implemented for XSEL. The message generator used to answer why-choice queries may be called a comparative message generator; it examines and compares the rank elements produced by the evaluation functions to determine what roles they play in the selection of the chosen component, and then it creates appropriate messages. The message generators used to answer the why-fact and why-line-item queries may be called link-dependent message generators; they examine the facts and the links between facts to determine what relations hold among them, and then they create appropriate messages.

Explanations produced by both the comparative message generator and the link-dependent message generators are certain to be accurate because they always originate from the contents of working memory. Special steps had to be taken to ensure the directness of the link-dependent message generators, however. Those steps will be discussed in the following sections, which describe the workings of the link-dependent message generators in some detail. Discussion of the comparative message generator and the surface generator will be reserved for other occasions.

3. Link-dependent Message Generation

3.1. Generic vs. Relational Explanations

Both of the link-dependent message generators are capable of operating in two modes, generic mode and relational mode. (The user can toggle between modes by typing "explain generic" or "explain relational".) The explanations shown above in Figures 1-4 and 1-5 are relational explanations; they explicate the relations that hold between facts. Some of those relations are arithmetic relations, such as sum and product, and some are abstract relations, such as satisfaction and allocation relations. Contrast the relational explanation for the query "why q total-disk-space" shown in Figure 3-1 with the generic explanation for the same query shown in Figure 1-4. Generic explanations do not explicate the relations that hold between facts; they simply state that some generic dependencies exist. The same message generator is used to generate both generic and relational explanations. (Notice that the same queuing mechanism is used to explain subsequent terms in both generic and relational explanations.) The difference between generic and relational explanations results from the fact that there are two different types of links in XSEL's memory, generic links and relational links. Both types of links establish a connection between two or more facts. The difference is that generic links are always unnamed, binary links, whereas relational links are always named, n-ary links, where the name may be an arithmetic relation such as sum or product, or an abstract relation, such as satisfaction or allocation. Both types of links are deposited into

```
? why q total-disk-space

THE VALUE OF 3600 MEGABYTES FOR TOTAL-DISK
IS DEPENDENT
ON 1424 KILOBYTES FOR TOTAL-APPLICATION-DIS
110592 KILOBYTES FOR PROGRAMMER-DISK-SPAC
2816000 KILOBYTES FOR TOTAL-DATA-FILE-DISK-S
600 KILOBYTES FOR PAGE-AND-SWAP-SPACE
AND 25600 KILOBYTES FOR SYSTEM-DISK-SPACE .

THE VALUE OF 25600 KILOBYTES FOR SYSTEM-DIS
IS DEPENDENT
ON VMS FOR OPERATING-SYSTEM .

THE VALUE OF 600 KILOBYTES FOR PAGE-AND-SW
IS DEPENDENT
ON 200 KILOBYTES FOR CODE-SIZE .

THE VALUE OF 2816000 KILOBYTES FOR TOTAL-DA
IS DEPENDENT
ON 2816000 KILOBYTES FOR DATA-FILE-DISK-SPAC

THE VALUE OF 110592 KILOBYTES FOR PROGRAM
IS DEPENDENT
ON 2048 KILOBYTES FOR EDITOR-DISK-SIZE ,
2816000 KILOBYTES FOR LARGEST-DATA-FILE ,
4 PROGRAMMERS FOR NUMBER-OF-PROGRAMME
AND 102400 KILOBYTES FOR LANGUAGE-USE-DISK

THE VALUE OF 1424 KILOBYTES FOR TOTAL-APPLI
IS DEPENDENT
ON 1024 KILOBYTES FOR SOFTWARE-DEDICATED-
AND 150 KILOBYTES FOR APPLICATION-DISK-SPAC
```

Figure 3-1: Sample Generic Explanation

XSEL's working memory by the reasoning rules that fire during program execution. As links are deposited during XSEL's execution, two dynamically growing networks are built up: the generic network is a simple dependency network, and the relational network is an augmented semantic network. These networks are the main source of knowledge for the link-dependent message generators.

A generic link is a very simple memory element consisting of only two attributes, a source attribute and a sink attribute. The value of the source attribute is the token (i.e., unique identifier) of some fact that entered into the inference of the resultant fact; the value of the sink attribute is the token of the resultant fact. For example, the rules that fire to infer a value for the fact total-disk-space will deposit into working memory at least five generic links, each having the token of the fact total-disk-space in its sink attribute and each having the token of a fact that entered into the calculation of the value for total-disk-space, such as total-application-disk-space, programmer-disk-space, etc., in its source attribute. An example of a generic link is shown in Figure 3-2. A relational link is a slightly richer memory element which not only names the relation that holds between two or more facts, but also categorizes it. Figure 3-3 displays one arithmetic relational link and one abstract relation link.

```
(generic-link
    ↑source <total-application-disk-space-token>
    ↑sink <total-disk-space-token>
)
```

Figure 3-2: Sample Generic Link

```
(relational-link
    ↑relation sum
    ↑category arithmetic
    ↑sink <total-disk-space-token>
    ↑source1 <total-user-disk-space-token>
    ↑source2 <sum-of-system-disk-space-token>
    ↑source3 <sum-of-page-and-swap-space-token>
)

(relational-link
    ↑relation satisfaction
    ↑category reason
    ↑sink <quantity-of-disks-token>
    ↑source <total-disk-space-token>
)
```

Figure 3-3: Sample Arithmetic and Abstract Relational Links

The network formed by relational links is in some places more dense and in other places less dense than the network formed by generic links; arithmetic relational links create more levels thus making the relational network denser, while abstract links tend to bridge long chains of facts, thus making the network sparser. To see this distinction, consider the arithmetic formula used by XSEL to calculate the total-disk-space requirement:

```
total-disk-space =
    ((total-application-disk-space
     + programmer-disk-space
     + total-data-file-disk-space)
     * 125% )
     + sum of system-disk-space
     + sum of page-and-swap-space
```

The rules that execute this formula create at least five generic links linking total-disk-space to total-application-disk-space, programmer-disk-space, total-data-file-disk-space, one or more system-disk-space facts, and one or more page-and-swap-space facts. At the same time they create one relational link linking total-disk-space to three new intermediate level facts, total-user-disk-space, sum-of-system-disk-space, and sum-of-page-and-swap-space, and they create additional relational links linking each of the intermediate facts to their subfacts. Total-user-disk-space is a newly created intermediate fact, and a relational link, with rrelation percent, is created linking it to two more new intermediate facts, user-disk-space and percent-for-expansion. Another relational link is in turn created linking user-disk-space to the three facts total-application-disk-space, programmer-disk-space, and total-data-file-disk-space.

On the other hand, the rules that determine how many RA60 disk drives are needed, for example, create a dense generic network linking all the facts that enter into the calculation of total-disk-space to the facts that allocate some portion of that amount to fixed-disk-space. From there the network would get even denser as fixed-disk-space is linked to the fixed-disk-unit-capability and quantity-of-fixed-disks facts for each candidate. In fact, these generic links are not currently created due to limitations of working memory space. In contrast to the potentially dense generic network, the relational network contains only a few abstract relation links, such as satisfaction and allocation links, that bridge many of the generic links, thus resulting in a sparser network (and in more direct explanations).

There are good reasons for the existence of two complete networks. Essentially, the tradeoff is that while generic links are trivial to create, they do not facilitate satisfying explanations. On the other hand, the creation of relational links often requires manual intervention, but relational links facilitate direct explanations. Compare again the generic explanation in Figure 3-1 to its corresponding relational explanation in Figure 1-4.

Generic links require little effort to create because they simply incorporate the tokens of the facts that are used in an inference rule. In fact, an automatic rule generator was developed for automatically creating most of XSEL's backward-chaining fact-gathering rules from simple arithmetic formulas such as the formula for total-disk-space discussed above.[1] It was a trivial task to have the automatic rule generator include the actions required to have the inference rules create the generic links.

The task of augmenting the fact-gathering rules to create arithmetic relational links was also automatable, for the most part. An automatic link-creator was written to parse the arithmetic formulas that were input to the rule generator and create the appropriate links. This parser identified the main arithmetic operations, created names for intermediate facts, and modified XSEL's rules to have them create the arithmetic relational links. The output of the automatic link-creator required only minor manual retouching in those cases where its heuristics for creating names for intermediate facts fell short.[2] But the task of augmenting the component selection rules to create the abstract relational links between facts has so far resisted an automatic solution. These links are now being added manually. They require the effort of someone who understands the workings of XSEL and recognizes what explanations might be called for and, consequently, which rules should be modified to create relational links.

3.2. Overview of Processing

The processing of a query by a link-dependent message generator goes as follows. When the initial query is input, a query-interpretation context is entered. In this context some rules fire to identify and locate the fact in question, to create a query-term with the same token as the fact, and to place that query-term in the query-queue. Following query-interpretation, a message generation cycle consisting roughly of the following five steps reiterates: 1) focus on the next query-term in the queue, 2) locate the links related to that query-term, 3) select an explanation schema[3] based on the links found, 4) create

[1] XSEL's automatic rule generator was written by Sandy Marcus.

[2] XSEL's automatic link-creator was written by Michael Browne.

additional query-terms and messages suggested by the selected schema, and 5) turn control over to the surface generator. Each time a new query-term is created, queue-control rules decide whether to place it in the query-queue, depending on such factors as whether the term has already been explained and how many levels of explanation the user has requested. As long as the query-queue is not empty, the message generation cycle is reiterated.

When the message generator is in generic mode, it is constrained to locating generic links during step 2 of the cycle, and it is constrained to selecting the generic schema during step 3 of the cycle. A simplified version of the generic schema is depicted in Figure 3-4. The first directive of the generic schema

```
(Schema-directives::Generic-schema
    (make goal ↑goal-name create-extra-query-terms
            ↑status reiterate)
    (make goal ↑goal-name create-message
            ↑predicate IS-DEPENDENT
            ↑term1 <current-focus>)
    (make goal ↑goal-name create-message
            ↑predicate ON
            ↑term1 <link-focus>
            ↑status reiterate)
)
```

Figure 3-4: The Generic Schema

directs the message generator to create additional query-terms for all the facts that are linked to the current query-term. The second directive directs the message generator to create one message with the predicate "IS-DEPENDENT" and with the focus-token of term1, which is the current query-term. The surface realization of this message will be the clause "THE VALUE OF 3600 MEGABYTES FOR TOTAL-DISK-SPACE IS DEPENDENT ". The third directive of the generic schema directs the message generator to create one additional message with the predicate "ON" and the focus-token of term1 for each of the link terms found. These messages will emerge as prepositional phrases in their surface form, such as " ON 1424 KILOBYTES FOR TOTAL-APPLICATION-DISK-SPACE , 110592 KILOBYTES FOR PROGRAMMER-DISK-SPACE , 2816000 KILOBYTES FOR TOTAL-DATA-FILE-DISK-SPACE , 600 KILOBYTES FOR PAGE-AND-SWAP-SPACE AND 25600 KILOBYTES FOR SYSTEM-DISK-SPACE ."

When the message generator is in relational mode, it is constrained to locating relational links and using relational schemas. There are a variety of each. Currently, relational links are categorized as being either reasons, elaborations, or arithmetic links. During step 2 of the message-generation cycle, the message generator searches first for reason links, next for elaboration links, and finally for arithmetic links. In some cases, the search for arithmetic links may be suppressed. For example, some links whose relation is allocation are subcategorized as being arithmetic operations, as in "75 percent of the total-disk-space requirement was allocated to removable-pack disks". In these cases, expressing the arithmetic relation also would be redundant.

When a relational link is located, a corresponding schema is selected. In contrast to the single generic schema, there are a variety of arithmetic and abstract relational schemas. Figure 3-5 illustrates the arithmetic "plus" schema that was used to generate the messages for the first paragraph of the "why quantity total-disk-space" relational explanation shown in Figure 1-4. It contains five directives, one to create the new query-terms found in the arithmetic reasoning trace and four to create messages. The second message creation directive will create as many messages as are needed to account for at least 80 percent of the total value of the fact being explained. (The 80 percent factor was implemented in order to filter out insignificant facts, thus making the explanation more concise. Another process that contributes to more readable explanations is the conversion of all units in different clauses of the explanation to the same highest common denominator, eg. megabytes.) Following that, two additional messages will be created, one to mention that the remainder of the total is accounted for by other terms and another to give an example.

Figure 3-6 illustrates the "satisfaction" schema that was used

[3] The term *schema* was adopted from the work of McKeown [11], who uses similar structures for discourse organization.

```
(Schema-directives:plus-schema
    (make goal ↑goal-name create-extra-query-terms
            ↑status reiterate)
    (make goal ↑goal-name create-message
            ↑focus-token <token1>
            ↑predicate CAPACITY-REQUIREMENT
            ↑subname RECOMMENDED)
    (make goal ↑goal-name create-message
            ↑focus-token new
            ↑predicate CAPACITY-REQUIREMENT
            ↑subname GENERAL
            ↑amount 80)
    (make goal ↑goal-name create-message
            ↑predicate REMAINDER)
    (make goal ↑goal-name create-message ↑focus-token new
            ↑predicate EXAMPLE)
)
```

Figure 3-5: Sample Arithmetic Schema

to create the messages for the first sentence of the "why quantity RA60" explanation shown in Figure 1-5. It contains one directive to create an extra query-term matching the token of the new term identified in the "satisfaction" link, and three actions making the three messages which surface as three clauses of text in the explanation.

4. Rationale

The knowledge structures just described, including messages, query-terms, the query-queue, schemas and links, serve as intermediate structures between the reasoning knowledge of the expert system and the linguistic knowledge needed for language generation.[4] Some of the terminology used to describe these structures, e.g., "reason" and "elaboration" relations, is derived from the work of Mann [7] and Hobbs [3] on discourse organization. Mann and Hobbs independently postulate that discourse relations, such as reason and elaboration relations among others, are responsible for coherence in well-organized

```
(Schema-directives:satisfy-schema
    (make goal ↑goal-name create-extra-query-term
            ↑focus-token <term2>)
    (make goal ↑goal-name create-message
            ↑predicate QUANTITY-SELECTED
            ↑term1 <term1>)
    (make goal ↑goal-name create-message
            ↑predicate INORDER
            ↑mtype relational-prop)
    (make goal ↑goal-name create-message
            ↑predicate CAPACITY-REQUIREMENT
            ↑subname SATISFY
            ↑term2 <term2>)
)
```

Figure 3-6: Sample Satisfaction Schema

natural language text. One of the premises of this work on explanation generation is that the relations, or links, that are embodied in the inference rules of a successful reasoning system are the same ones that give coherence to natural language explanations. An immediate goal of this research is to identify those relations. At the present time only twenty-six different reasoning relations, have been identified in XSEL. As more types of reasoning relations are identified and corresponding links are added to XSEL's rules, more of XSEL's reasoning will be explainable. A long term goal of this work is to continue to identify and add reasoning links and schemas until we see some generalities begin to emerge. Perhaps some domain-independent set of reasoning relations and schemas might be found. Furthermore, such relations and schemas might facilitate the design of a knowledge acquisition system that would elicit knowledge from an expert, represent it as relations, and generate inference rules from relations. We realize that this could be a *very* long term goal, but it also has the short term benefit of providing useful explanations.

[4] See Mauldin [9] for another view of intermediate knowledge structures needed for language generation

Acknowledgements

Many people at CMU and DEC have contributed to the development of XSEL. Some of these include John McDermott, Tianran Wang, and Kim Smith who developed XSEL's sizing and selection knowledge; Robert Schnelbach and Michael Browne who worked on explanation facilities; Sandy Marcus, who wrote XSEL's rule generator; George Wood, Jim Park, and Mike Hannon who provided technical support; and Dan Offutt who is extending XSEL's sizing knowledge with a view towards developing knowledge acquisition facilities.

References

1. R. Davis. *Applications of meta level knowledge to the construction, maintenance, and use of large knowledge bases.* Ph.D. Th., Stanford University, 1976. Stanford Artificial Intelligence Laboratory Memo 283, Stanford, CA.

2. C. L. Forgy. OPS-5 User's Manual. CMU-CS-81-135, Dept of Computer Science, Carnegie-Mellon University, Pittsburgh, PA 15213, July 1981.

3. Jerry R. Hobbs. Towards an Understanding of Coherence in Discourse. In W. G. Lehnert and M. H. Ringle, Ed., *Strategies for Natural Language Processing*, Lawrence Erlbaum Associates, New Jersey, 1982, pp. 223-243.

4. Gary Kahn, Steve Nowlan, and John McDermott. A Foundation for Knowledge Acquisition. Proceedings of the IEEE Workshop on Principles of Knowledge-Based Systems, IEEE, Denver, CO, 1984, pp. .

5. Gary Kahn and David Geller. MEX: An OPS-based approach to explanation. 1984.

6. Karen Kukich, John McDermott and Tianran Wang. XSEL as Knowledge Acquirer and Explainer. 1985.

7. William C. Mann and Sandra A. Thompson. Relational Propositions in Discourse. 1983.

8. Sandra L. Marcus, John McDermott and Tianran Wang. A Knowledge Acquisition System for VT. Proceedings of the AAAI, AAAI, Los Angeles, CA, 1985, pp. .

9. Michael Mauldin. Semantic Rule Based Text Generation. Proceedings of the 10th International Conference on Computational Linguistics, ACL, Stanford University, Stanford, CA, 2-6 July 1984, pp. 376-380.

10. John McDermott. Building Expert Systems. Proceedings of the 1983 NYU Symposium on Artificial Intelligence Applications for Business, New York University, New York City, April 1983.

11. Kathleen Rose McKeown. *Generating Natural Language Text in Response to Questions about Database Structure.* Ph.D. Th., University of Pennsylvania Computer and Information Science Department, 1982.

12. William R. Swartout. "XPLAIN: a System for Creating and Explaining Expert Consulting Programs". *Artificial Intelligence 21* (1983), 285-325.