MASSACHUSETTS INSTITUTE OF TECHNOLOGY

ARTIFICIAL INTELLIGENCE LABORATORY

Working Paper 178 JANUARY 1979

THE XPRT DESCRIPTION SYSTEM

LUC STEELS

ABSTRACT

This paper introduces a frame-based description language and studies methods for reasoning about problems using knowledge expressed in the language.
The system is based on the metaphor of a society of communicating experts and incorporates within this framework most of the currently known AI techniques, such as pattern-directed invocation, explicit control of reasoning, propagation of constraints, dependency recording, context mechanisms, message passing, conflict resolution, default reasoning, etc.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. The author was sponsored by the Institute of International Education on an ITT-fellowship.

CONTENTS

INTRODUCTION

A *description system* is a method for interacting with computational systems which is based on the ability to express knowledge about a certain domain in the form of descriptions. When the user has a certain problem, he communicates this problem in the form of a description again. The system has the capacity of interpreting these descriptions and of solving the problem by reasoning over its available body of knowledge.

For example, suppose we are interested in geometrical objects, then we would introduce concepts like triangle, line, angle, distance, etc. to our description system. Also we would tell it about the constraints among these concepts, such as the fact that a triangle has three lines which are in a particular relation to each other. Based on this knowledge a description system should be able to solve a variety of tasks such as recognize an object as being a triangle (because it satisfies all constraints), construct a triangle given initial specifications, deduce properties of triangles, and so on.

Description systems contain a *description language* for expressing constraints on a set of situations or objects.
When faced with a particular task, the system will first of all build up a finite network of objects and relationships starting from a set of initial constraints or premises given by the task and using the general constraints it knows about. Such a network will be called a *constraint network*
Several things can be done once the network is there. On the one hand we can extract the results we are interested in. But even more interesting applications are imaginable. It should be possible to change the initial constraints or default assumptions that were made during the build-up of the network and have the effects of these changes being propagated through the network so as to keep the constraint relationships valid. This allows us to do some experimentation or keep track of changes in the environment. Also it should be possible to change the body of knowledge (i.e. the constraints itself) and propagate these changes through the network.
In this paper we will concentrate especially on the process of constructing constraint networks.

At present several description systems have been proposed and implemented. See e.g. the constraint language of Sussman and Steele (1978), a system called ThingLab constructed by Alan Borning (1979), the description language of Hewitt, et.al. (1979), a.o. Description systems have been shown to be useful in the simulation and investigation (analysis and synthesis) of physical systems like electronic circuits. (cf.Sussman and Steele, ibid.) This could easily be extended to other systems, e.g. biological ,organisms, the weather, programs, etc. In fact, there is great use for description systems everywhere where design and recognition tasks are involved. (Think for

example about the design of a house).
We are applying these methods to natural language processing. In this application the constraints represent constraints on linguistic objects (i.e. constitute a grammar) and constraint networks are constructed in performing linguistic tasks: When a sentence is observed, the initial observations and existing expectations serve as the initial constraints or premises and by consulting the grammar the network is expanded until it contains a complete description of the communication situation. Similarly in production, the goals of the communication form the initial constraints and a constraint network is being created that satisfies these initial constraints and the constraints of the grammar. (More about all this in Steels (1978b,1979)).

We believe that description systems are the next step in attempts to bridge the continuum between *explicit programming* which is based on 'the ability to specify and control actions down to the first detail' (Hewitt,1972,22) and *implicit programming*, which presupposes 'the ability to specify the end result desired and not to say much about how it should be achieved' (o.c).

PLANNER-like systems (Hewitt (1969), McDermott and Sussman (1973), Davies (1973),a.o.) can be viewed as the first step in this direction. These systems have built-in mechanisms to perform functions considered to be basic for intelligent processing. The mechanisms concentrate mostly on providing relief on the *procedural* aspects of intelligent systems: the maintenance of databases, pattern directed invocation of rules, the maintenance and manipulation of contexts, etc. By providing these basic mechanisms the task of constructing intelligent systems becomes more tractable.

FRAME-based knowledge representation languages (Minsky (1974), Roberts and Goldstein (1977), Bobrow and Winograd (1977),etc.) and semantic network formalisms (Brachman (1976), Hendrix (1975), Norman and Rumelhart (1973), etc.) can be viewed as another step. These systems concentrate on the problem of *representing* knowledge. To this purpose they provide the user with a number of primitive data structures, selector and construction routines for these data structures, etc. As such they bring relief on the problems of representing large amounts of information in an epistemologically adequate way.

In the present system we will try to make a synthesis of these two developments. We will use valuable ideas of PLANNER-like systems and valuable ideas of frame-based knowledge representation languages and semantic networks. The metaphor of a society of communicating experts will act as a unifying framework.

In the paper we swing back and forth between the introduction of the description language (which was introduced in Steels (1978a)) and a discussion of the way the reasoning goes. On the whole we remain on an intuitive level. Other papers concentrate on philosophical

foundations, formal semantics, exact characterization of the procedural primitives, etc. This paper is more an introduction to our approach.

1. FRAMES AND DESCRIPTIONS

We start from the idea that knowledge about a certain subject matter should be grouped together. Such a grouping will be called a *frame* (following Minsky,1974). A frame contains all the constraints that are known to hold between the conceptual *aspects* of a certain concept.

We will set up a frame structure as follows

```
(<frame-name>
   (WITH <aspect-name-1> )
   (WITH <aspect-name-2> ) ...  )
```

For example

```
(LINE
   (WITH BEGIN)
   (WITH END)
   (WITH LENGTH) ...)
```

sets up a frame for LINE with conceptual aspects for the begin, the end, the length, etc. Each of these aspects introduces a so-called *slot*.

From a (formal) semantic point of view, a frame represents a set of possible configurations of individuals that may fill the slots of a frame. For example if we have a frame for SUM:

```
(SUM
   (WITH RESULT)
   (WITH ADDEND)
   (WITH AUGEND))
```

then the interpretation of this frame is a set of triples of numbers which are in the SUM-relation. Each configuration in this set is called an *instantiation*.

Frames are the basis for descriptions. *Descriptions* do two things: they isolate one particular instantiation of a frame and they focus on the individual that plays a particular role (called the *view*) in that instantiation. Descriptions are represented in terms of list-structures as follows

```
    (<view> <frame-name>
            (WITH <aspect-name-1> )
            ...
            (WITH <aspect-name-n>))
```

as in

```
    (RESULT SUM
            (WITH ADDEND)
            (WITH AUGEND))
```

which can be read as 'the result of a sum with a certain addend and a certain augend'. This description picks out an instantiation from the instantiation-set of the SUM-frame (i.e. a particular triple of numbers) and focuses on the element that plays the RESULT role in this instantiation. RESULT is the view of the description.

We will use descriptions to specify constraints on the individuals that can fill certain slots in frames or descriptions. Attaching a description to a slot expresses the information that every individual which is a filler of the slot to which the description is attached, is/has to be the filler of a slot (named by te view) in the instantiation pointed at by the description. Attachment is represented by writing the description after the name of the aspect introducing the slot.
For example suppose that we have a frame for number, as in

```
(NUMBER
  (WITH VALUE))
```

and that we construct a description from this frame, as in

```
  (VALUE NUMBER)
```

i.e. 'the value of a number', then we can attach this description to a slot in the SUM-frame as follows

```
(SUM
  (WITH RESULT
        (VALUE NUMBER))
  (WITH ADDEND)
  (WITH AUGEND))
```

This frame now contains the information that every individual which is the result of a SUM-frame plays the VALUE role in an instantiation of NUMBER. In short that every result of a sum is the value of a number.

An attached description may function like a *selection restriction* on what kind of things may fill a certain slot. It may also act like a *consequent* when we know that an object is the filler of the slot.
Notice that, as a consequence of our definitions, descriptions *inherit* the descriptions attached to the aspects in the frame used in that description. For example once we know that the value of TWO is the result of a SUM, we know that it is the value of a NUMBER.
Notice also that we can construct *frame hierarchies* by using a description of the more general concept as constraint on the more specific one, as in

```
(REAL-NUMBER
  (WITH VALUE (VALUE NUMBER)))
```

It happens that we want to specify more complex relations between several slots, i.e. not just a relation between one slot (the view) of a description and a slot in a frame but a relation between a pair of slots in one instantation and a pair of slots in another instantiation. To allow for this capability we have to introduce a new type of specifying constraints. This new type is known as a *co-referential description*.

Here is an example. Suppose we have the concept of a parent-child-relation as in

```
(PARENT-CHILD-RELATION
  (WITH PARENT)
  (WITH CHILD))
```

and the concept of MOTHER-CHILD-RELATION with aspects for the mother and the child as in

```
(MOTHER-CHILD-RELATION
  (WITH MOTHER)
  (WITH CHILD)).
```

Now we want to specify that the pair mother-child corresponds to the pair parent-child. Note that something like

```
(MOTHER-CHILD-RELATION
  (WITH MOTHER
        (PARENT PARENT-CHILD-RELATION))
  (WITH CHILD
        (CHILD PARENT-CHILD-RELATION))).
```

is not a sufficient constraint because it does not say that the mother is the parent of the same instantiation of the parent-child-relation as the child is the child of.

What we will do is say that the child slot of the parent-child-relation in the description attached to the mother slot is co-referential with the child slot in the mother-child-relation.
Co-referential links are represented by writing '(= <unique-name>)' after each slot that is co-referentially related. The unique name is lexically scoped within one frame.
For the MOTHER-CHILD-RELATION frame this leads us to

```
(MOTHER-CHILD-RELATION
  (WITH MOTHER
        (PARENT PARENT-CHILD-RELATION
                (WITH CHILD (= THE-CHILD))))
  (WITH CHILD
        (= THE-CHILD)))
```

Before introducing more complex representational constructs we discuss activation methods for the representational constructs introduced so far.

2. CONSTRAINT NETWORKS.

We will use the metaphor of a *society of communicating experts* as a source of ideas for constructing the reasoning system. (See Minsky and Papert (forthcoming) and Hewitt(1976) for work in a similar direction.)

We define an *expert* to be an active object that has a body of knowledge about a particular subject matter and a script.
The *script* specifies how the expert should behave: how it should respond to requests for information about the knowledge it is responsible for, how it should try to expand incoming descriptions by asking questions to other experts, how it should maintain consistency of the descriptions by preventing the introduction of contradictions, etc.
Both the body of knowledge and the script are dynamic entities: they can change and grow depending on the functioning of the whole system and the structure of the tasks they have to deal with.

Each expert follows its own course of action. The experts operate in *parallel*, although each expert processes messages 'one at a time' to avoid synchronization problems. Moreover an expert can not treat another expert as an object. The only interaction that can take place is by *message passing*. In short an expert is a particular sort of actor. (cf. Hewitt,1976)
Each expert can only communicate with a limited number of other experts: it can communicate with itself, with the expert that was responsible for its creation (the so called ancestor) and with experts it knows the name of. We call the experts that an expert can communicate with its *acquaintances*.

Experts start off as copies of *prototypical experts*. This happens when a prototypical expert receives a request to solve a particular problem and rather than working on this problem, it creates a copy and starts this copy on the given task.
When there is a collection of experts such that each expert is a copy of the same prototypical expert and each expert is able to communicate with another expert of the same group, we call the collection a *society*. The acquaintance relationships give structure to the society.

In this paper we will study two types of experts: frame-keepers and object-carriers.
Frame-keepers are experts that have as body of knowledge a single frame. The major purpose of the script is to respond to requests for information about the frame. A society of frame-keepers for a particular domain of knowledge is organized as a tangled generalization hierarchy.
Object-carriers are experts that have a collection of descriptions which are true for a particular object in the domain. The script contains mainly methods to expand descriptions by tracking down their consequences. Part of this expansion is the creation of new object-

carriers that start reasoning over objects related to the object the expert is reasoning about. A society of object-carriers that is working on a particular problem is called a *constraint network*.

It is clear that there are many other types of experts that could be investigated or that the function of the experts mentioned could be substantially expanded. For example it is conceivable to extend the script of the frame-keepers with rules that would perform retrieval-functions by propagating markers (as in Fahlman,1978), with rules that would perform learning tasks, etc. What has been designed and implemented so far is only the beginning.

We see then that a constraint network consists of a collection or society of experts called object-carriers which exchange messages with other experts in the same society or with frame-keepers. Each object-carrier is reasoning about a certain (anonymous) individual in the domain of discourse.

An object-carrier will need some sort of memory to keep track of what descriptions it already received. To this purpose we will view the collection of descriptions in an object-carrier as a data-base of patterns. The script of an object-carrier is then viewed as a PLANNER-like pattern-directed invocation system (see AMORD (De Kleer,et.al. 1978) for a recent example). Each rule in this system consists of a trigger and an action. When the trigger matches with an incoming message, the action is performed. The action might result in adding a new description to the database, in requesting more information from other experts, in sending descriptions to other object-carriers, etc.

For our present discussion we will let a pattern consist of an expression of the description language preceded by a control indicator. When such a pattern is sent as message to a certain expert we will write this as

<name-of-the-expert> <- [<control-indicator> <description>]

For example

XPRT-1 <- [given (VALUE NUMBER)]

tells the expert called XPRT-1 that it is the value of a number.

INSTANTIATION

The first thing an object-carrier does when it receives such a message is perform an *instantiation*. It creates new object-carriers for each of the (anonymous) objects that are known to be part of the instantiation introduced by the description. (Refinements follow later). Each of these new experts receives a message specifying what its role is in the new instantiation. We refer to experts by a 'semi-description': (CALLED <expert-name>). Also the new experts receive the descriptions that were originally attached to the slot in the source-description.

For example, suppose that we have a frame for MINUS-ONE:

```
(MINUS-ONE
  (WITH ARGUMENT)
  (WITH RESULT))
```

and that we send to an object-carrier the following description:

```
XPRT-1 <- [given (RESULT MINUS-ONE
                         (WITH ARGUMENT
                               (VALUE ONE)))]
```

then an instantiation would result in the creation of a new object-carrier, further called XPRT-2, and in the exchange of the following messages:

```
XPRT-1 <- [known (RESULT MINUS-ONE
                         (WITH ARGUMENT
                               (CALLED XPRT-2)))]
XPRT-2 <- [given (VALUE ONE)]
XPRT-2 <- [known (ARGUMENT MINUS-ONE
                           (WITH RESULT (CALLED XPRT-1)))]
XPRT-2 <- [known (VALUE ONE)]
```

Note how XPRT-2 received the description that was attached to the argument-slot in the instantiated description. Note also that the control-indicator is changed from given to known for each description that has been instantiated. Only 'known' descriptions will be valid in further reasoning.

A refinement of this method of instantiation will enable us to deal with *partial descriptions*. A partial description is a description that has not all of the aspects which the frame used in the description has. For example

```
   (RESULT MINUS-ONE)
```

is a partial description because the ARGUMENT-aspect is missing.

An object-carrier instantiates partial descriptions by asking to the frame-keeper of the frame used in the description what the 'ideal' set of aspects are. Then it creates new experts for each of those aspects and sends them the descriptions which are attached to the corresponding aspect in the original description. If the aspect does not occur in the description, the expert receives no other descriptions.

PROPAGATION OF CONSTRAINTS

Now suppose that an object-carrier has a certain description of an object. So it knows that this object plays a certain role (the view of the description) in an instantiation of the frame specified in the description. The next thing to do then is ask the frame-keeper of the frame used in the description whether it knows any additional constraints on this object. The frame-keeper will reply with the description that is attached to the aspect which is the view of the description. This constraint then enters the object-carrier and becomes part of the global description of the object. We call the activity of distributing the constraints that are attached to the

aspect of a frame *propagation of constraints*.

Here is a simple example. Suppose we have the frame for MINUS-ONE again and a frame for NUMBER as follows

```
(NUMBER
  (WITH VALUE)).
```

Furthermore suppose that we attach constraints to the slots in the MINUS-ONE frame:

```
(MINUS-ONE
  (WITH ARGUMENT
        (VALUE NUMBER))
  (WITH RESULT
        (VALUE NUMBER))).
```

Then if we send to an expert the following partial description

```
  XPRT-3 <- [given (RESULT MINUS-ONE)]
```

an instantiation will occur, leading to

```
  XPRT-4 <- [known (VALUE MINUS-ONE
                          (WITH RESULT
                              (CALLED XPRT-3)))]

  XPRT-3 <- [known (RESULT MINUS-ONE
                          (WITH VALUE
                              (CALLED XPRT-4)))].
```

Now we propagate the constraints from the MINUS-ONE frame, leading to

```
  XPRT-3 <- [given (VALUE NUMBER)]
  XPRT-4 <- [given (VALUE NUMBER)]
```

The result is that the objects over which the two experts are reasoning have been further restricted to objects which are the value of a NUMBER. This new description will again be instantiated and may lead to further constraints, etc.

CO-REFERENTIAL LINKS

Recall that the description language allows for the specification of so called co-referential links between slot-fillers. An important aspect of propagating constraints consists therefore in distributing the knowledge that certain object-carriers are thinking about the same individual.

We will first look at the simplest case and discuss more difficult cases later. The simplest case assumes that it is possible to construct a local environment in the process of instantiation, which contains bindings in the form of explicit references to certain experts for each of the indirect references in a frame. When a frame is instantiated or a constraint is passed to an object-carrier, we replace the indirect-references with their bindings.

An example will make clear what is going on here. Let us take the MOTHER-CHILD-RELATION frame introduced earlier. Assume that the following initial description is sent to an object-carrier

```
XPRT-5 <- [given (MOTHER MOTHER-CHILD-RELATION
                         (WITH CHILD (BEING MARY)))]
```

instantiation will lead to

```
XPRT-6 <- [known (CHILD MOTHER-CHILD-RELATION
                        (WITH MOTHER (CALLED XPRT-5)))]
XPRT-5 <- [known (MOTHER MOTHER-CHILD-RELATION
                         (WITH CHILD (CALLED XPRT-6)))]
XPRT-6 <- [given (BEING MARY)]
```

While performing this instantiation we also created a local environment relating the name 'THE-CHILD' with XPRT-6.
Now we propagate the constraint attached to the MOTHER-aspect in the MOTHER-CHILD-RELATION frame and take care to replace the indirect references by their bindings:

```
XPRT-5 <- [given (PARENT PARENT-CHILD-RELATION
                         (WITH CHILD (CALLED XPRT-6)))]
```

Further instantiation of this description will lead to

```
XPRT-6 <- [known (CHILD PARENT-CHILD-RELATION
                        (WITH PARENT (CALLED XPRT-5)))]
XPRT-5 <- [known (PARENT PARENT-CHILD-RELATION
                         (WITH CHILD (CALLED XPRT-6)))]
```

Note that we passed by another refinement of the instantiation method. When there is already an expert reasoning over a certain object (this is the case if the view of the description is CALLED), the process of instantiation will not lead to the construction of a new expert. We will simply take the given expert and send it all the relevant descriptions. This happened here with the instantiation of the parent-child-relation frame. The child was filled by a pointer to an expert (i.e. XPRT-6) so this expert is then used in the instantiated description.

3. MERGING DESCRIPTIONS AND KEEPING TRACK OF IDENTITY LINKS

It is clear from the foregoing discussion that we should try to make every effort to keep the number of object-carriers as limited as possible. (The minimum is of course one expert for each object we are reasoning about - but that turns out to be an unattainable goal). We now introduce further refinements of the instantiation method with this purpose in mind. First we extend the description language with information that is particulary valuable in this context.

We often observe interesting properties on the set of possible instantiations of a frame. The most interesting example of such a property is the following: When a certain individual fills a particular aspect in an instantiation of a given frame, there is no other instantiation of that frame where the same individual fills the same aspect. We say then that this aspect is *criterial* in that frame.
Criteriality is generalized over more than one aspect as follows. When a certain series of individuals is known to fill a corresponding series of aspects in a certain instantiation of a given frame, there is no other instantiation of that frame where the same series of individuals fills the same series of aspects.

In the frame of LINE given earlier, BEGIN and END are criterial because there are no two lines with the same begin and the same end. In the same frame the BEING aspect in itself is also criterial. But the BEGIN aspect on its own is not criterial, because there can be two lines with the same begin.
It is clear that the characterization of criterial aspects is similar to determining the identity-conditions of a concept.

We will represent this information as follows. We distinguish between two parts of the frame: the frame-structure and the aspect-specification. The frame-structure contains the aspects and the constraints on the aspects. The aspect-specifications is a list of specifications of the form (<specification> <list-1> ... <list-n>) where a list contains the aspects that satisfy the specification.
For the LINE frame this leads us to

```
(LINE
  (FRAME-STRUCTURE:
     (WITH BEGIN)
     (WITH END)
     (WITH DISTANCE)
     (WITH BEING))
  (ASPECT-SPECIFICATIONS:
     (CRITERIAL:  (BEGIN END) (BEGIN DISTANCE)
                  (END DISTANCE) (BEING))))
```

The default for criteriality is non-criterial. If we are not interested in the aspect-specifications we do not write this more elaborate structure but use the simple form introduced earlier.

What is the importance of knowing what aspects are criterial? Its importance is precisely that it enables us to be more selective with creating new object-carriers during instantiation. Two examples will illustrate this point.

Suppose we have the following frame

```
(FATHER-CHILD-RELATION
  (FRAME-STRUCTURE:
    (WITH FATHER
          (PARENT PARENT-CHILD-RELATION
                  (WITH CHILD (= THE-CHILD))))
    (WITH CHILD
          (= THE-CHILD)))
  (ASPECT-SPECIFICATIONS:
    (CRITERIAL:  (CHILD))))
```

In other words a child can only have one father.
Now suppose that there is an XPRT-1 who received the following description

```
XPRT-1 <- [known (CHILD FATHER-CHILD-RELATION
                        (WITH FATHER (CALLED XPRT-2)))]
```

and that it then receives the following message

```
XPRT-1 <- [given (CHILD FATHER-CHILD-RELATION
                        (WITH FATHER (HAS-NAME JOHN)))]
```

then we do not have to create a new instantiation of the FATHER-CHILD-RELATION frame but we can use the existing instantiation and send attached descriptions (e.g. '(HAS-NAME JOHN)') to the expert which is reasoning over the object filling the corresponding slot. We say in this case that the incoming description is *merged with* the existing one. This action is justified because an individual can only once be the child of a father-child-relation. Therefore the father of the first instantiation has to be identical with the father of the instantiation we would have constructed for the second instantiation.

Here is a second example how knowledge of criteriality leads to more control over constructing the constraint network. It could be that the description which is attached to a particular slot refers to an instantiation of which the respective objects are already present in the network. So we could make an attempt to discover the names of the object-carriers reasoning over the objects before making attempts to create new object-carriers as we would do according to the simple scheme proposed in the previous section.

Let us continue with the example just given. Suppose there is a third object-carrier (XPRT-3) which receives the description

```
XPRT-3 <- [given (WIFE MARRIAGE
                       (WITH MAN
                             (FATHER FATHER-CHILD-RELATION
                                     (WITH CHILD (CALLED XPRT-1))))))].
```

XPRT-3 will now attempt to instantiate this description. Because CHILD

is a criterial aspect in the father-child-relation frame, it looks whether it can find the name of the expert reasoning over the father by asking XPRT-1 whether its object is described as the child of a father-child-relation. Indeed there is such a description and it is discovered that XPRT-2 is already reasoning over the father. As a result we can use XPRT-2 as the expert reasoning over the filler of MAN.

Further refinements are possible and necessary. For example we can sometimes distinguish instantiation-groups within the set of possible instantiations of a frame. Within such a group there are certain aspects that are single-valued but other aspects which are not. We call these aspects *projective* (following Hewitt). When we merge two descriptions, only those aspects which are projective are effectively merged. This point will not further be explored here.
Another refinement that will not be explored has to do with so called *individuating* aspects which allow for the construction of individual descriptions. An individuating aspect is an aspect that can only once be filled by an individual. When an instantiation is performed, we will know immediately what kind of expert to use for the filler of this aspect because we keep track of a list of individuals which have individual descriptions.

The criteriality declarations help substantially in keeping down the creation of unnecessary object-carriers. But it raises other problems. Suppose the description contained already a reference to a particular expert and suppose we find out that there is another description that can be merged with the new description. This other description contains of course also a reference to a unique expert and it could be a different one. What really happens here is that the system realizes that two distinct object-carriers are actually reasoning over the same individual !
What we do in such a case is establish an *identity-link* between the two object-carriers which makes them virtually identical, in other words if a message is sent to one of the object-carriers, the script of both object-carriers looks at it.

There are some other issues that remain to be explored here, but the general line of inquiry should be clear now. Let us therefore turn to some interesting expansions of the fundamental power of the description language.

4. CONDITIONALS

Conditionals refer to a more complex constraint, namely one where it is known that there is a relation between the instantiation sets of two frames but this relation only holds if the instantiation we are looking at is related to another instantiation. We say then that the first is conditionally related to the second.

For example the parent of a parent-child-relation is the mother of a mother-child-relation if the individual filling up the parent slot in the instantiation of the parent-child-relation is at the same time related to an instantiation of the female-person frame.
In other words the relation between the parent-child-relation frame and the mother-child-relation frame is conditionally related to the female-person frame.

We will express conditional links as follows. First we introduce the entity for which the conditional holds by using the indirect-reference occurring elsewhere in a co-referential description. Then we give a list of pairs where the first element is the condition that the entity has to satisfy in order for the second element to be a valuable description.

All this is represented as

```
(WHEN <the-referring-name>
      (<condition-1> <resulting-description-1>)
      ...
      (<condition-n> <resulting-description-n>))
```

meaning that as soon as a condition is known to hold for the object indicated by the referring-name, the corresponding resulting-description is known to hold. The last condition could be the symbol 'ELSE', which stands for the conjunction of the negation of the other conditions.

For example

```
(WHEN THE-PARENT
      ((BEING FEMALE-PERSON) ;  then
       (MOTHER MOTHER-CHILD-RELATION
               (WITH CHILD (= THE-CHILD))))
      ((BEING MALE-PERSON)
       (FATHER FATHER-CHILD-RELATION
               (WITH CHILD (= THE-CHILD)))))
```

So when the individual which occurs as interpretation of THE-PARENT is known to be the filler of the being slot in an instantiation of the female-person frame, the resulting description is

```
  (MOTHER MOTHER-CHILD-RELATION
          (WITH CHILD (= THE-CHILD)))
```

Conditional expressions may be attached to slots in the same way as descriptions. The idea is that the descriptive relation holds between

the individual filling the slot and the description which would result from resolving the conditional.

So we have

```
(PARENT PARENT-CHILD-RELATION
        (WITH PARENT (= THE-PARENT)
              (WHEN THE-PARENT
                    ((BEING FEMALE-PERSON) ;  then
                     (MOTHER MOTHER-CHILD-RELATION
                             (WITH CHILD (= THE-CHILD))))
                    ((BEING MALE-PERSON)
                     (FATHER FATHER-CHILD-RELATION
                             (WITH CHILD (= THE-CHILD))))))
        (WITH CHILD (= THE-CHILD)))
```

A special case of the conditional is the IFF (if and only if):

```
(IFF <the-referring-name>
     (<condition-1> <resulting-description-1>)
     ...
     (<condition-n> <resulting-description-n>))
```

This conditional expresses the information that if the resulting description holds for the slot-filler to which this description is attached, then we know that the condition-1 holds and if a condition holds for the individual named by the-referring-name, we know that the corresponding resulting description holds.

Let us now turn to the problem of reasoning with these conditionals. The framework sketched earlier allows for a straight forward incorporation.

What we do is send a pattern-directed invocation rule to the expert that is reasoning about the object referred to by the referring-name for each of the lines in the conditional. The trigger of the rule corresponds to the condition in the line. The action corresponds to sending the expert reasoning over the slot-filler the corresponding resulting-description. All this has the effect that as soon as the condition is satisfied the resulting description will end up in the appropriate expert.
When the conditional is an IFF, we send in addition a pattern-directed invocation rule to the expert that is reasoning about the slot-filler with as trigger the resulting condition and as action the sending of the corresponding condition to the expert reasoning over the object pointed at by the referring-name.

Conditionals are a very important tool in controlling how the network will be expanded. For example when a hierarchy is identified in the domain, one could incorporate 'downward' pointers in the form of conditional expressions. So, rather than saying concept A specializes into concept B, C and D, we say concept A specializes into B if such

and such a condition is satisfied, C if another condition is satisfied, etc. Based on this information the reasoner can make a well-founded decision on which things to explore further, instead of performing some sort of search that explores all possibilities.

5. THE CONNECTIVES

We would like to have the ability to combine descriptions with connectives such as AND, OR, XOR and NOT. Syntactically we represent such combinations as follows

(<connective> <description-1> ... <description-n>).

The problem is not how to represent or interpret connectives but how to reason effectively with descriptions containing connectives. We will follow the methods of natural deduction (cf. Kalish and Montague, 1972) using the techniques of 'explicit control of reasoning' (DeKleer, et.al., 1977). The scripts of the object-carriers will have rules that decompose descriptions or create new rules trying to satisfy subgoals. Here are some examples.

When the argument of a negation is another complex description we will transform this description in order to bring the NOT inside the connective, using well known inference rules from propositional calculus.
For example if we have

```
(NOT
  (AND (VALUE 2)
       (VALUE 3)))
```

a pattern-directed invocation rule will turn this description into

```
(OR
  (NOT (VALUE 2))
  (NOT (VALUE 3)))
```

making good use of DeMorgan's law.

When an AND-combination of a number of descriptions as in

```
(AND (BEING FEMALE-PERSON)
     (MOTHER MOTHER-CHILD-RELATION
             (WITH CHILD (BEING MARY))))
```

enters an object-carrier, we can decompose this description by adding each of the conjuncts as new descriptions.

Things become more difficult if we consider other connectives. What we would like to do with an XOR-combination of descriptions (let us consider the case of two for simplicity) such as

```
(XOR (FATHER FATHER-CHILD-RELATION
             (WITH CHILD (BEING JOHN)))
     (MOTHER MOTHER-CHILD-RELATION
             (WITH CHILD (BEING JOHN))))
```

is set up two contexts. One in which the first description holds and the negation of the second description, and one in which the second description holds and the negation of the first. When we continue reasoning, we will keep track of the context in which the antecedents occurred. Further results, if any, are context-bound. Thus we are able to perform a parallel *case-analysis* of the various possibilities.

We now introduce additional mechanisms to do just that. The mechanisms

are (i) associating context-declarations to each description in an expert (a technique developed in early pattern-directed invocation systems (cf.McDermott and Sussman (1973), Rulifson et.al.(1973), a.o.)) and (ii) keeping track of justifications describing the logical dependencies between decriptions (a technique developed in Sussman and Stallman (1975) and further refined in the so called truth-maintenance systems of Doyle (1978) and McAllester (1978)). We will see later that there are other applications of the same mechanisms.

In the present system a *context* is essentially a collection of descriptions that is grouped together for a particular reason. For example we could have a context containing all descriptions that are true at a certain moment of time.
Contexts are related to each other forming so called *context-structures*. A context-structure relates contexts by making use of a *context-transformer* which contains a description of the difference between the two contexts involved. Typical aspects of this transformer are whether the first context is a specialization of the other one, i.e. whether all descriptions valid in the first context are inherited from the second one, what facts are no longer valid in the other context, etc.

When an expert looks at a certain description (for example to see whether it matches with a trigger of a rule, or to instantiate it) it first of all checks whether the description was asserted in the context it is working in. If not it tries to find out whether the description can be transported from other contexts into the current context using the context-transformer. If this is the case a note is made that the description is valid in the new context. If not the description is ignored.
It now becomes clear why we need to keep track of the dependencies: When a certain fact depends upon the validity of other facts it can only be transported to the new context if all of its antecedents are valid in this new context.

Note that an expert only considers the effect of a context-change when it is interested in a certain fact. This is different from existing truth-maintenance systems which sweep through the database at the moment when a conflict occurs. Our method is necessary if parallelism is maintained as basic mode of operation. A second difference with existing truth-maintenance systems is that we step through contexts rather than update continuously the status of facts in a single context. This gives us additional flexibility which will be explored later.

We can now deal with XOR based on this context-mechanism. The script of the expert will contain a pattern-directed invocation rule that will be triggered when an XOR-description comes in. Then two contexts are created which are sub-contexts of the currently existing one. Within the first context we assert the first part of the description and the negation of the second. In the second context we assert the second

part of the description and the negation of the first. Reasoning continues and if a deduction is performed which uses a fact in one of these contexts, this context-restriction is recorded as part of the new fact.

These techniques apply also to the treatment of descriptions with OR such as

```
(OR (FATHER FATHER-CHILD-RELATION
            (WITH CHILD (BEING JOHN)))
    (MOTHER MOTHER-CHILD-RELATION
            (WITH CHILD (BEING JOHN))))
```

Now an expert has to set up three contexts. One context in which both components are valid, one context in which the first is valid and the negation of the second and a third one in which the second description and the negation of the first is valid.

An entirely different class of composition mechanisms relates to the use of complex descriptions in conditionals. Faced with such a description, we let an expert construct more complex rules.

For example when a conjunction of descriptions occurs as condition in a conditional the object-carrier will create a complex pattern-directed invocation rule that will first look at the first conjunct and if this is found will create a new rule that looks at the second conjunct, and so on until all descriptions are found. Only then the resulting-description will be sent to the appropriate expert.

With an exclusive or, several mechanisms are possible. For example we could set up a rule looking at the first component and if this is found set up a rule looking at the negation of the second one. Only if the latter is found the resulting description is triggered. At the same time we could look at the negation of the first component and if this is found set up a rule looking at the second component. Only if the latter is found the resulting description is sent out. An alternative mechanism would again start looking for one component, but rather than waiting for the satisfaction of the other component it would assume the negation of the other component and send out the resulting-description.

With an OR-combination of descriptions as condition, the expert will set up a series of rules where each rule has one of the disjuncts as trigger. As soon as such a rule is satisfied the resulting-description will be sent to the appropriate expert.

The techniques introduced in this section prove to be effective for dealing with expressions with connectives. We now turn to more elaborate reasoning strategies.

6. DEFAULTS AND PREMISE-CHANGES

It is now generally accepted that not everything can be known by a system at all times. So assumptions have to be made in order to make certain necessary decisions. When the assumptions later on turn to be unjustified, they can be retracted and a different course of action can be taken. Also it has been recognized that often it is possible to specify a 'most likely case' which would receive preference in such cases of uncertainty. This most likely case is called the default-case.

We will now extend our description language in order to give default-specifications. We will do this in the context of the conditional with the following syntax:

```
(UNLESS <referring-name>
        (<condition-1> <resulting-description-1>)
        ...
        (ELSE <default>))
```

meaning that unless one of the conditions is found to be valid in which case the corresponding resulting-description is asserted, the default will be assumed. So we are not only able to specify a default but also the conditions under which a default should be retracted.

Our framework provides a straight-forward way to make this construct effective. What an expert will do is first ask to the expert responsible for the object denoted by the 'referring-name' whether any of the conditions are satisfied. If one of them is, the resulting-description is sent out, just like with normal conditionals. However if none of them is satisfied, the default is sent out and rules are set up that keep looking out for the conditions. As soon as one of those conditions becomes valid (e.g. due to user-input), the default is being retracted and the new resulting-description is sent out.

By 'being retracted' we mean that a new context is being created which is related to the previous one as a sub-context but differs in that the default-assumption is no longer valid. The original context is discarded as 'not the one that is valid at the present moment' and questions or further deductions will reconsider the validity of facts before using them as antecedents. We see therefore that the context-mechanism and the depedendency-recording which we introduced for dealing with the connectives is useful in this application too.

We are now only one step from another powerful application: the change of premises, which happens when certain facts are invalidated (e.g. because results from sensory data come in or because the user wants to experiment with the constraint network.). What we want to see happen is that the constraint relations imposed by the frames on the various descriptions are maintained.

Again the context and dependency mechanisms come to a help here. The

change of original assumptions will be considered to be a change of the context in which observations were made. When facts are touched, they will be reconsidered in this new context and the change of premises may cause the triggering of rules which were looking out for certain descriptions.

7. CONCEPTS OF HIGHER LEVEL

Consider now the following representation problem. We want to say about a relation (e.g. the subset-relation) that it is a transitive relation. This amounts to saying that if the subset-relation holds between the superset of a subset-relation and another set, the subset-relation also holds between the subset of the first relation and this other set.
We could of course represent this information explicitly in the subset-relation frame as follows (leaving out descriptions defining the normal constraints on the slots in the subset-relation)

```
(SUBSET-RELATION
  (WITH SUPERSET (= THE-SUPERSET))
  (WITH SUBSET
        (WHEN THE-SUPERSET
              ((SUBSET SUBSET-RELATION
                       (WITH SUPERSET (= A-THIRD-SET)))
               (SUBSET SUBSET-RELATION
                       (WITH SUPERSET (= A-THIRD-SET)))))))
```

But that is not really what we want. We want to introduce a frame for transitive relations and simply say that the subset-relation is a transitive relation. 'Transitive relation' is a concept of second level because it specifies a property of concepts. This can be generalized to still higher levels in an obvious way. What we need in order to make all this concrete is a way of specifying mappings from the frame-name of one frame to the frame-name of another one and from the aspects of a frame to the aspects of another frame. We will do this by introducing a *viewed-as* operator denoted by / (following Moore and Newell,1974).

Returning to our example, we first define the transitive relation

```
(TRANSITIVE-RELATION
 (WITH ARG1
       (WHEN THE-SECOND-ARGUMENT
             ((ARG1 TRANSITIVE-RELATION
                    (WITH ARG2 (= A-THIRD-ARGUMENT)))
              (ARG1 TRANSITIVE-RELATION
                    (WITH ARG2 (= A-THIRD-ARGUMENT))))))
 (WITH ARG2
       (= THE-SECOND-ARGUMENT)))
```

and then we have as subset-relation frame

```
(SUBSET-RELATION
 (WITH SUBSET
       (ARG1/SUBSET TRANSITIVE-RELATION/SUBSET-RELATION
                    (WITH ARG2/SUPERSET (= THE-SUPERSET))))
 (WITH SUPERSET
       (= THE-SUPERSET)))
```

The description attached to the subset slot of the subset-relation frame can be paraphrased as "the filler of the subset slot is the first argument viewed as subset aspect of a transitive relation viewed as

subset-relation such that the second argument viewed as superset aspect is co-referential with the filler of the superset slot".

We can deal in a straightforward manner with higher level concepts by having a rule that performs a manipulation of the descriptions in the higher level concept frame. This rule replaces the aspect and frame-names with the names which are mapped onto it by the the viewed-as operator.

For example, suppose we have the following situation

```
XPRT-1 <- [known (SUBSET SUBSET-RELATION
                         (WITH SUPERSET
                               (CALLED XPRT-2)))]
XPRT-2 <- [known (SUBSET SUBSET-RELATION
                         (WITH SUPERSET
                               (CALLED XPRT-3)))]
```

Now we try to resolve the WHEN-description attached to the ARG1 in the transitive-relation frame. In principle we would send a rule to th script of XPRT-2 looking out for

```
(ARG1 TRANSITIVE-RELATION
      (WITH ARG2 (= A-THIRD-ARGUMENT)))
```

However with the viewed-as operator this becomes

```
(SUBSET SUBSET-RELATION
        (WITH SUPERSET (= A-THIRD-ARGUMENT)))
```

This matches with one of the descriptions in XPRT-2 and we send the following resulting description which is again transformed to accomodate the VIEWED-AS specification to XPRT-1:

```
(SUBSET SUBSET-RELATION
        (WITH SUPERSET (CALLED XPRT-3)))
```

8. CONCLUSIONS

Although the discussion remained on an intuitive level, it will be clear that the framework sketched in this paper successfully combines the methods of frame-based knowledge representation with the methods of pattern-directed invocation systems. The resulting system is more powerful in two respects. In other frame-based systems, the user has to write special purpose activation methods for each of the actions that he wants to see performed. In the present system the user only has to specify the declarative aspects of the frame. Activation is performed by the reasoner which knows how to turn the representations into active objects.
The advantage as regards pure pattern-directed invocation system which use simple patterns as basic tool of representing information, consists in the fact that we can structure our knowledge and use descriptions instead of simple patterns. Moreover because of the declarative nature of the knowledge representation, the same knowledge structures can be used for many different purposes. This is not the case if the knowledge is explicitly encoded in terms of rules.

There are many aspects that we did not cover in this paper. The most important one being the handling of conflicts. An object-carrier is constantly attempting to keep its database consistent. When an inconsistency is noticed a message is sent to a *complaint-department* that tries to bring relief in one way or another. This relief can come from censors which repress certain outcomes or the development of certain lines of thought. Conflicts could also be the basis for triggering frame-shift mechanisms which would adjust the constraints in a frame to accomodate for new situations.

9. REFERENCES

Bobrow,D. and T. Winograd (1977) An overview of KRL- A Knowledge Representation Language. 3-46 in Cogntive Science, 1.1.

Borning, A. (1977) ThingLab -- An Object-Oriented System for Building Simulations Using Constraints. 497-498 in IJCAI-4, Cambridge Ma.

Brachman (1976) What's in a concept. COLING-76. Ottawa.

Davies, et.al. (1973) POPLER: A POP-2 PLANNER. MIP-89. University of Edinburgh.

De Kleer, J. et.al. (1977) Explicit Control of Reasoning. MIT-AI Memo.

Doyle, J. (1978) Truth-maintenance systems for problem solving. MIT-AI TR-419.

Fahlman, S. (1977) A System for Representing and Using Real World Knowledge. MIT-AI TR-450.

Hendrix, G. (1978) Partitioned Semantic Networks.

Hewitt, C. (1969) PLANNER: A Language for Manipulating Models and Proving Theorems in a Robot. IJCAI-69. Washington, D.C.

Hewitt, C. (1972) Description and Theoretical analysis (using schemata) of PLANNER: A language for proving theorems and manipulating models in a robot. Ph.D. thesis MIT.

Hewitt, C. (1975) How to use what you know. MIT-AI WP 93

Hewitt, C. (1976) Viewing control structures as patterns of passing messages. MIT-AI memo

Hewitt, C., G. Attardi and H. Lieberman (1979) Specifying and Proving properties of guardians for distributed systems. MIT-AI WP.

Kalish and Montague (1973) Techniques of formal reasoning. Chicago: University of Chicago Press.

McAllester, D. (1978) Three-valued truth maintenance. MIT-AI Memo 473.

McDermott and Sussman (1973) Why conniving is better than planning. MIT-AI memo 25

Minsky, M. (1974) A framework for representing knowledge. in Winston (ed.) The Psychology of Computer Vision. New York. Mc Graw Hill. 1975.

Minsky, M. and S. Papert (forthcoming) The society of mind theory.

Moore and Newell. (1973) How does MERLIN understand ? in Gregg (ed.) Explorations in Cognition. Potamac, Md: Lawrence and Erlbaum Ass.

Norman and Rumelhart (1973) Explorations in Cognition. San Francisco: Freeman.

Roberts, and I. Goldstein (1977) The FRL Manual. MIT-AI Memo. 409

Rulifson, Johns, Derksen and Waldinger. (1973) QA4: A Procedural Calculus for Intuitive Reasoning. Ph.D. Stanford.

Sussman, G. and G. Steele (1978) Constraints. MIT-AI Memo 567.

Stallman, and Sussman, G. (1977) Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis. Artificial Intelligence 9. 135-196.

Steels, L. (1978a) Frame-based Knowledge Representation. MIT-AI WP 170.

Steels, L. (1978b) Some examples of conceptual grammar. MIT-AI WP 177.

Steels, L. (1979) Grammatical Dependencies. in preparation.

10. Acknowledgment

Communications with Beppe Attardi, Roger Duffy, Carl Hewitt, Henry Lieberman, Gerald Sussman and Marvin Minsky have helped to give shape to the ideas contained in this paper.