Working Paper No. 140

A BIRTHDAY PARTY FRAME SYSTEM[1]

by Gregory D. Clemenson
Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Febuary 1977

ABSTRACT

This paper is an experimental investigation of the utility of the MIT-AI frames system. Using this system, a birthday party planning system was written, representing the basic decisions that comprise such a plan as frames. The planning problem is presented in the user in a way conforming to his natural planning procedures. The system is able to check the consistency of the plan parts, and finally produces a completed plan for the party, and can supply the user with some valuable summaries, such as a shopping list.

[1] This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-75-C-0643.

S.D.G.

CONTENTS

A BIRTHDAY PARTY FRAME SYSTEM

The purpose of this paper is evaluate the utility of "frame" systems in facilitating the task of planning a birthday party. This particular context has been chosen because it is one of the settings to which frames were originally applied [Minsky 1975]. The birthday party setting was originally used to introduce frames because it contained certain problems that the frames idea could handle in an elegant fashion. The questions then are how elegantly the frames concept is capable of handling problems in an implemented system, and why frames perform at that level.

PROBLEM ANALYSIS

Planning birthday parties is basically the same as planning anything else. While the details of plans differ, there are identifiable, common techniques and organizational principles that are used in all successful planning. We shall attempt to identify these and apply them in a carefully executed example, birthday party planning, using a frame based implementation. There are actually three issues here. First there is the frames system itself, built with an eye towards this kind of planning. Then there is our analysis of the planning process, and of the birthday party planning in particular, and last there is the actual implementation. Here we shall briefly discuss these issues.

Fundamentally, frames are packets of data that conveniently organize all of the knowledge about a specific item. They are concise statements about the essence of a particular item. Most of the lower level details are pushed off into other frames, which then can be referenced as sub-frames. They serve to organize all of the pertinent parts and relationships, without the details necessarily being there. They facilitate focussing on one particular principle or thought. Each such nexus is intended to be more than just an organizational component, or a convenient data template. Its parts, viz. slots, contain not just values or pointers to other frames, but also information and procedures that tell explicitly how to deal with the associated values, e.g. how to make new ones, what to do if one should be changed or deleted, etc. The MIT-AI frames system is discussed in detail in Goldstein and Roberts [1977]. Below is an example of a frame in this system. It will be discussed later.

```
(frame  SEND-INVITATIONS
        (ako         ($value       (giving)))
        (recipients  ($if-needed   ((get-guest-list))))
        (donor       ($if-needed   ((get-party-host))))
        (medium      ($default     (send-by-mail)))
        (gift        ($require     ((ako? :value 'card))))
        (message     ($if-needed   ((get-invite-message)))
                     ($if-added    ((write-on-cards)))))
```

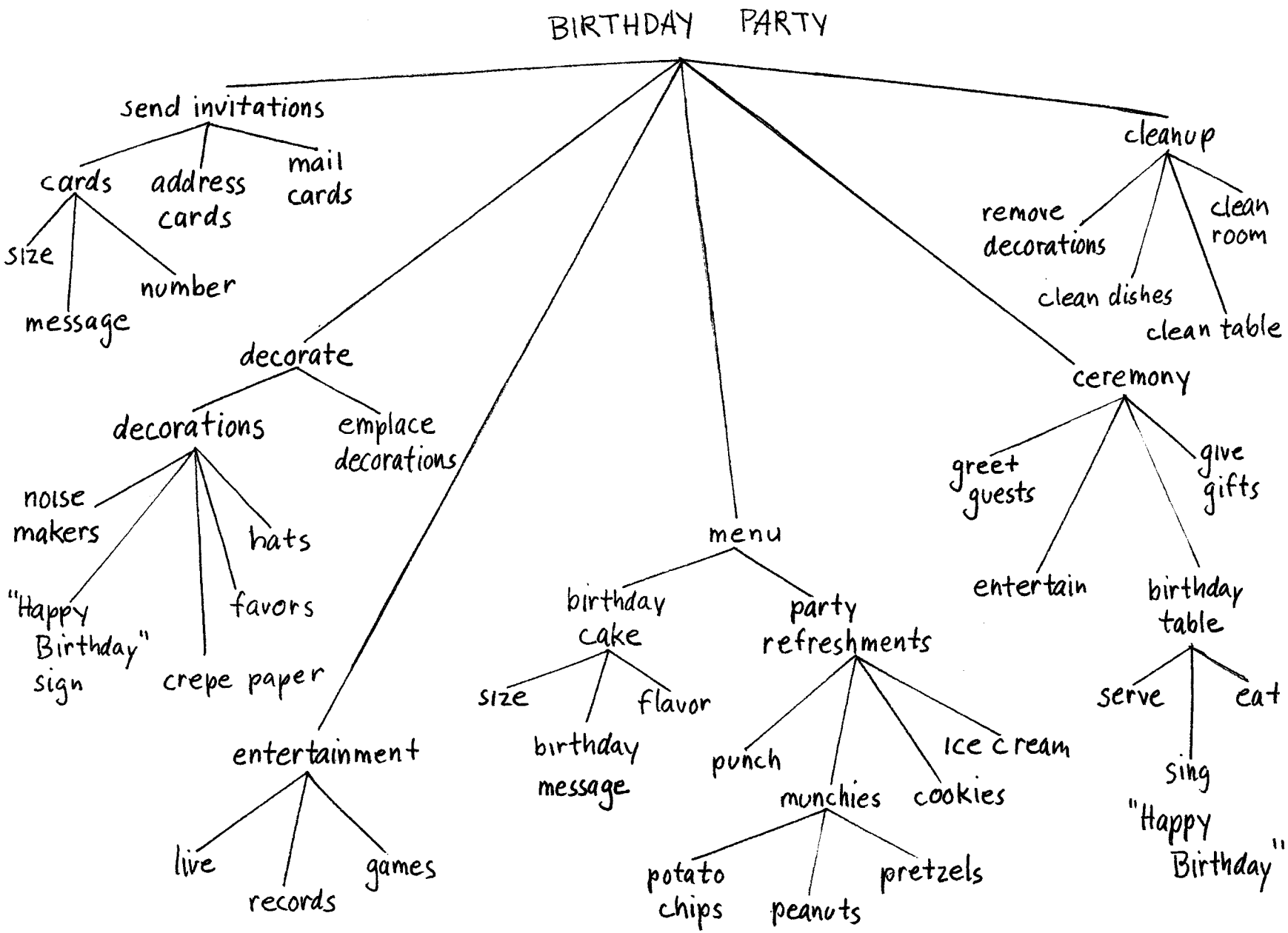
Frames are the tool, but here they are useful only in that they are capable of embodying the planning process. So we must see what this process actually entails before is cast in the form of a working frame system. We will look at planning in the context of planning an activity, such as a birthday party. There is a standard approach to planning a birthday party, so we will not be involved in selecting basic approaches and implementations, an aspect of planning more fully developed in Miller and Goldstein [76]. The fact that the planner has decided to have a "birthday party" means that he has already agreed to a basic kind of plan. Even a little variation may be enough to make the same "birthday party" inappropiate. What variations there are in birthday parties represent all known decisions and alternatives at various points in the plan. For example, the refreshments can vary widely, as can the decor and extent of the decorating. Thus our main concern is the organization and taxonomy of these standard sorts of plans and the manner in which the planner wishes to encounter their parts.

The goal of such planning is to ultimately produce some sort of procedure for actually having a birthday party. We want an exact plan that would tell exactly who was to do what at what time, as well as possibly outlining some sort of alternatives that may be to be put into use if the need arises. It is not convenient for any user to create a plan the serial sort of manner in which it is to be executed. The problem is taking care of all the preconditions and results of each part of the plan. An example is shopping. In order to prepare a shopping list, the user must know everything that each part of his plan might need from the store. This includes various decorations, food, invitation cards, disposable eating utensils, cleaning fluids, and what ever else might be required. Shopping is one of the first things done, but planners must first know most of the rest of the plan before they know what to buy. Thus it might be said that all plans are not equal. Users will prefer to work on what they consider to be the major items first, and the details and interactions of these later.

Our taxonomy of plans is primarily molded by these considerations. We are going to characterize a plan as being a representation of one particular collection of decisions to be presented to the user. The decisions consist of specifying the particular details of the standard plan, such as the name of the person who is having a birthday, and specifying the choices that are a part of the standard plan, such as what kind of decorations to use. There are both subjective and objective criteria for these groupings. Subjectively, these groupings are chosen with regard to what a planner might think are the major decisions that occur at a given level of planning. For example, choosing the menu, that is the snacks and refreshments to be served at the party, seems to be a coherent group of related decisions. But this does not seem to be on quite the same level as specifying a punch recipe. The recipe is a sub-part. Concepts such as part-whole relationships, and superior-inferior procedures are the objective criteria for choosing groupings. The important thing to note is how aspects of one plan affect another. If the plan is altered at some point, the parts that are affected most are likely to be the sub-parts. On the next page is a picture of how we have organized the birthday party plan, along such a line of sub-plans and brother-plans.

In our implementation each on of these nodes is represented as a frame. Each of the sub-parts has a particular relationship to its superior. These relationships are the slots. The interactions between the various nodes are captured by procedures attached to the frames. These can be regarded as information that describes what the effect is on the rest of the plan if their particular slot is created, altered, or removed. The process of specifying the plan parts, and dynamically checking these interactions, is what finally creates the finished birthday party plan. While in this form, it is easy to show the user what the effects of choices are, and to explain the interactions to the user in a lucid manner. When all of the choices have been specified, and the interactions have been established, we have a structure that can be easily turned into a sequential program.

The system as outlined so far, cannot handle all of the interactions. The problem is that there may be global effects that purely local information cannot solve, much like the classic three block problem that Sussman's Hacker program attacked [Sussman 1975]. Our mechanisms can only handle what he calls a linear plan. Since the user is really in control of the planning, it is not necessary to have such a global planning solver. The only place where such problems occur is with time. For example, if one has three activities, A, B, and C. Suppose that B must be after A, and before C. We may not know explicitly that this means that A must precede C. In fact, we can schedule A after C, and only get into trouble when we try to schedule B. What would be nice is some system awareness of this problem so that the user can be told what is going on.



Outline of Birthday Party Frame System Organization

AT THE TERMINAL

In this section we will look at a session with an implemented frame based birthday party system. "C" prefaces the computer's typed messages and output. Everything that the user types is prefaced with a "U". Other remarks that merely present further information or explanation, but which are not part of the actual user-computer dialogue, are not prefaced.

U: (schedule birthday-party)

The system requires that the user use the actual frame names to identify the frames to be used. At times the system may present the user with a list of lettered possibilities to fill in a frame slot. In such cases it is only necessary to specify the letter that cooresponds to the desired choice. It is hoped that such a frames system might eventually be able to handle natural language input. At this time the system has created an "instance" of the birthday party frame. That is Birthday-party already exists as a frame that contains information about how to set up and plan a birthday party. An instance of this is made to be a copy of this, only with all of the plan parts specified, and the planning decisions made. Of course, as this represents a particular party, and not birthday parties in general it may contain all of the options that the general one may have. Also, the user may want to add things that are not in the general plan, so in this sense it is not an exact copy. But since the user may not want to specify all of the details at once, everything need not be specified. These parts and details of birthday parties are represented by slots in their frames. Filling in the details coresponds to specifying values for these slots. So far, this birthday party instance has no details specified, so most of its slots are empty. In the absence of further information, the details default to specifications included in the general plan. These are represented as special information attached to the slots and are called "defaults".

C: Who is the birthday party for?

In order to set up a birthday party frame, the first thing that the system has to do is to determine the birthday-person. What has actually happened is that the birthday party frame has started to fill in the birthday-person slot, which in turn has asked the user for some reference to this person's frame. This question is recited by the computer to prompt the user, and let him know what the system wants. There are two possible outcomes: either

the name given by the user is already known by the system, in which case the computer assumes that it can get relevant information, such as the person's age from his frame; or the person is not known by the system (i.e. there is no person frame by that name in the system's data base), in which case the system will make a frame for him, and proceed to fill it out. Let's assume that this latter case occurs.

U: Larry

C: How old is Larry?

The system did not know of a person named Larry, so it instantiated a person frame with Larry as the person's name. The next step is to fill in the slots of the person frame for Larry, such as his age. Like the previous case, the computer is using a canned sentence, except that it does a kind of fill in the blank with the person's name. The system has by this time already defined a frame by the name Larry, and is just filling in the details. Using these kinds of questions, with possibly more elaborate fill in the blank sort of operations is the common way that such details get filled in these frames. Right now this is all of the information that is needed about Larry. So the system goes on in a similar sort of way, finding out about things such as the party-place, and the party's cost.

U: 16

C: Where will the party be held?

U: Roger's residence

The system understands this to be a reference to the residence slot in the Roger frame. It finds the Roger frame, and the residence slot has the value house37, the frame name for Roger's house. So the system adds house37 to the place slot of the birthday-party frame.

C: How many people will be at birthday-party4?

U: 40

C: Do you want a budget for birthday-party4?

U: Yes

C: How much money do you want to spend on birthday-party4?

U: 40.00

The system has now set up a special budget frame to keep track of expenses and will interrupt the user if the specified limit is exceeded.

C: These are necessary sub-parts of birthday-party4.
You may choose any of these to work on,
or $next to stop work on this slot,
or nil to add a new one.
A Decorate-bdparty12
B send-invitations13
C party-entertain14
D bdparty-menu15
E party-cleanup16
F bdparty-ceremony5

U: D

Since all of these choices are necessary, they have been automatically instantiated, with only a minimal filling in of their slots. The user is asked which one he wants to work on now, that is fill in further. He chooses birthday-party eating, whose main concern is the menu.

C: Do you want to have a birthday-cake?

U: Yes

In a similar sort of way the system instantiates a birthday-cake frame and proceeds to fill in the details, by asking the user leading questions.

C: What flavor should the cake be? The usual choices are:
A Vanilla-cake
B Chocolate-cake
C Banana-cake

```
U:  C

C:  What flavor should the icing be?  The usual choices are:
    A White-icing
    B Chocolate-icing
    C Pineapple-icing

U:  Mocha-icing

C:  What is the source for mocha-icing
    A Grocery-store
    B Recipe

U:  $quit
```

When the system saw mocha-icing, and realized that it did not know any frame by that name, it decided to make it a frame. The system was expecting a kind of icing. So mocha-icing was created as an instance of the frame icing. With any food item, it is necessary only to specify either where to get it (e.g. meat at the butcher's), or its recipe. The user decides to not bother with mocha-icing for the moment.

```
C:  What should birthday-cake6 say?  The usual is
    A "Happy Birthday Larry".

U:  Yes
```

The user could have written in any message here, but decided to use the system's choice.

```
C:  What kind of cake should it be?
    A Sheet-cake
    B Layer-cake

U:  B

C:  How many layers should it be

U:  3
```

C: How big should the cake be in inches?

U: 12

The user has completely specified the cake frame, so the system returns to deciding on menu items for bdparty-eating15.

C: What else do you want for bdparty-eating?
A Soda
B Punch
C Ice-cream
D Cookies

U: C

C: What flavor ice-cream do you want?
A Chocolate-icecream
B Vanilla-icecream
C Fudge-brownie-icecream

U: C

C: How many half-gallons do you want?

U: 10

Having completed this ice-cream frame, the system returns to the menu until the user specifies that he is done.

C: What else do you want for bdparty-eating15?
A soda
B Punch
C Ice cream
D Cookies

U: No

Since the menu is completed, the system has completed bdparty-eating15. The user may add to it later, and will be given the opportunity to do so whenever the system again presents the sub-plans of birthday-party4. The computer now returns to this sub-plan level, and the user indicates that he is done, so this session terminates. He can come back at any time to finish specifying the plan for the birthday party.

Several things have been happening that the user may be totally unaware of. The system has started a shopping frame that contains a shopping list of things that must be bought. It realizes that ice cream requires dishes and spoons, as well as refrigeration. So it has checked to see if house37 has these. Even though the user has not yet said anything about cleaning up, the system has already instantiated the cleanup frame, and told it to get sponges necessary to clean up the ice cream. It was necessary to do all of this in order to put some of the information that the system generated, in the form of implications for other parts of the plan, into their proper places. Also the budget frame has been quietly keeping track of all of the expenses

Later the user returns to continue specifying what he will do in birthday-party4. The function frame-update will allow the user to further specify or redefine any slot. In particular, the user still has much to do in specifying the details of the various sub-plans. He chooses to work on entertainment for the party, known to the system as the frame: party-entertain14.

```
C:  What kind of entertainment do you want?
    A Records
    B Live
    C Games

U:  B
```

The system now instantiates a live-entertainment frame, and proceeds to fill it in.

```
C:  Which group do you want?
    A Local-yokals
    B Crummy-mummies

U:  B
```

```
C:  You have exceeded your budget.
    The budget limit for birthday-party4 is $40.00.
    Below are the budget items.
    You may choose to modify one of these,
    or nil if the limit is to be ignored for now.
    A (birthday-cake20 $10.00)
    B (fudge-brownie-icecream21 $20.00)
    C (crummy-mummies $20.00)
```

The computer has been aware all along of the cost limitations of the party. The crummy-mummies was known to the system as a singing group that costs $20.00. Now that the budget has been exceeded, the user is given the choice of modifying the costly items. With some items, like ice-cream, cost can be reduced by reducing quantity. With others, the cost is fixed. The user can either update one of the slots in the item selected, or delete the references to the item. The system is slightly clever in deleting references to a frame that is currently being instantiated, as in the present case, and also stops the instantiation.

```
U:  C

C:  You may do either of these with crummy-mummies:
    A Frame-Update
    B Delete

U:  B

C:  What kind of entertainment do you want?
    A Records
    B Live
    C Games

U:  A
```

...the session continues.

Finally, the user asks to see the budget, and the shopping list.

```
C:  Here are the budget items:
    Birthday-cake20           $10.00
    Fudge-brownie-icecream21  $20.00
    Peanuts24                 $1.00
    Punch25                   $3.00
    Pretzels27                $1.00
    Cards30                   $5.00
    You have spent:           $40.00
    The budget limit was:     $40.00

    Here is your shopping list:
    Birthday-cake20           Bakery          1   Cake
    Fudge-brownie-icecream21  Baskin-Robbins  20  Pints
    Peanuts24                 Grocery-store   1   Pound
    Orange-juice26            Grocery-store   1   Can
    Ginger-ale27              Grocery-store   2   quarts
    Pretzels28                Grocery-store   1   Package
    Cards30                   Stationer       40  Cards
    Stamps31                  Post-office     40  $.09 stamps
```

HOW THE SYSTEM DOES IT

Frames are more than just templates that can be used to store data in a standardized format. Each slot of a frame can contain not only a value of some sort but special programs and information that specifies how the values of that slot are to be chosen, handled, and interpreted, etc. In fact, any kind of action or information that deals with these values can be put in the pertinent slot. This system uses the MIT-AI frame system [Goldstein and Roberts 1977]. In this birthday party system, every activity, and entity that the system uses is represented as a frame. The basic process of planning for a specific birthday party consists of specifying the precise objects and activities that each slot of the birthday party frame refers to. This specified copy of the general birthday party frame is given its own name, and refers to the specific birthday party being planned.

The slots of frames can also specify such information as how to get a value for the slot, and for supplying suggestions, defaults, and preferences. The to-add part of a slot is a program that is to be run to get a value for the slot. Information that can guide the user

can either be stored in this program directly, or the program can get it from the preferrences, suggestions, or defaults. Suggestions can include things that the system may not know about explicitly, such reference books, people, or magazines that could provide the user with useful information for filling this slot. At times it will be necessary to place a value in a slot even if the user declines or has no opportunity to specify the value. This system will usually use defaults for this purpose.

Generally, the system uses requirements (found under the $require key of the slot) to filter bad choices for slot fillers. For example, when asking for the number of guests at the party, only a number is acceptable. However, requirements can also supply information about the kinds of fillers that a slot can expect. In the case of the mocha icing, the icing slot of the cake frame has a requirement (ako? icing). When the user typed mocha-icing, the system did not know any frames by that name, and proceeded to create one. This simple kind of slot restriction can be used to supply information about the new frame, such as the fact that it should be a kind of icing. So a frame by the name of mocha-icing was created, with (ako ($val (icing))) as its ako slot. I suspect that this system is a bit too presumptious in trying to forge ahead with creating this new frame. The actions here would probably make a good default kind of action, but the user should be asked for confirmation, since he may have made a mistake.

Here is a specific example of some frames in the birthday party system.

```
(frame   GIVING
    (ako         ($value      (activity)))
    (gift        ($if-needed  ((ask [|What is being given?|]))))
    (recipients ($if-needed  ((ask [|Who should| @!gift |be given to?|]))))
    (donor       ($if-needed  ((ask [|Who is giving| @!gift |?|]))))
    (medium      ($if-needed  ((ask [|How is| @!gift |being given?|])))))
```

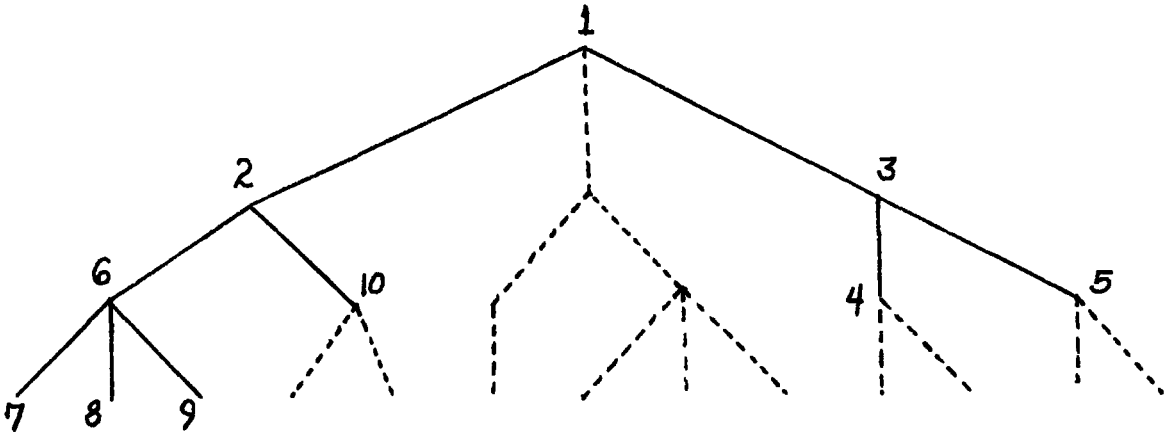
Giving is a kind of activity. In general, one can only tell what is being given, and to whom, etc. by asking the user. Ask is a program that does this. The strange "@!gift" notation is just the fill in the blank mechanism, that will put the name of the filler in the gift slot into the question that is typed as a prompt to the user. Send-invitations is just a kind of giving.

```
(frame   SEND-INVITATIONS
    (ako         ($value      (giving)))
    (recipients  ($if-needed  ((get-guest-list))))
    (donor       ($if-needed  ((get-party-host))))
    (medium      ($default    (send-by-mail)))
    (gift        ($require    ((ako? :value 'card))))
    (message     ($if-needed  ((get-invite-message)))
                 ($if-added   ((write-on-cards)))))
```

Here we know a little bit more about what is being given, and to whom, etc. The to-adds are more specific, and make use of the fact that the values of these slots refer to other frames in the system.

The system should present the user with only the choices that bear on the planning of a birthday party, and it should do it in an order more or less according to the user's wishes. To accomplish this, there are certain ways in which the frames system, as it existed, was changed. For one, the system too frequently presented the user with many detailed choices that he was not prepared to make, and which might have been very irrelevant. It was necessary to create a more flexible kind of frame-filling procedures. The desire is to be able to follow the user in as natural a way as possible, as outlined in the introduction. The user would generally tend to want to specify all of the related general choices, before plunging too much off into specifics. If the user were selecting the menu, it is conceivable that he would not desire to specify the punch ingredients, and then return to the menu selection process, but would rather decide on the complete menu, and then fill in the details. Utilities, such as the frame-update and slot-update were created to give the user as much control as possible in ordering the way in which things are planned. He can postpone choice presented to him by the system, or may, at his own choice plunge into the details of some plan. The sort of control over the planning process that the utility functions enable is outlined below. It is quite similar to the controllable search process created by Goldstein and Miller in their structured planning editor [1976].

We also wanted to create the possibility of a limited sort of instantiation, in which only a selected subset of the usually filled-in slots are filled. Ordinarily when a frame is instantiated, its self slot tells which slots to fill and in what order. Thus when a person frame is instantiated, the system ordinarily wants to know where he lives, his age, sex, etc. These may be quite irrelevant in a given context. For example, the user would probably not really want to give all of the birthday person's vital statistics when planning a birthday party. The person's age and possibly sex are all that are really necessary. This makes a



Possible Path of a User in Specifying a Plan

good system default that does not present the user with irrelevant choices. If the user wants to fill in further, he can use frame-update to do so.

If a particular slot in a frame is to be filled in with another frame, that slot might want to fill in a different set of slots from the slots indicated in the self slot of the instantiated frame. Slots can do this using special to-adds that tell which slots are to be filled. Some slots, like the ako slot, must always be filled in. Whatever planning method that seems natural at a given point can be constructed in the to-adds.

Frame-update and slot-update can also be controlled with the commands $n(ext) and $q(uit). The first one causes work to cease on the current slot, and starts the process of selecting a new slot to work on. $q causes work to cease on the current frame. Control then reverts to the program that called frame-update, so the effect, as far as where in the planning scheme you were in, is the same as if the frame had been completed.

With this sort of flexibility, rescheduling is really not too different from the standard frame filling process. Again, the user is presented with the unfilled slots, and is then given

the opportunity to further specify some slot. He may add to it or whatever. He also can continue to fill in one of the frames referred to in the slot. In this way, the user can work his way down the tree of frames, and work on any desired node.

The system has to be smart about some of the difficulties that can be caused by the user skipping around in his planning. For example, the user might not want to tell the system the birthday person's age when he is asked for it. But if the user later asks for a selection of games to choose, the program that generates the suggestions will generate different ones depending on what age the birthday person is. Rather than just not suggesting anything, or suggesting everything possible, the system can try to fill in this needed slot, and run the to-add for the person's age, fill in the slot with the returned value (if the user is cooperative), and use it to complete the suggesting task.

The cost monitoring feature is instituted by $if-added procedures. Any frame that costs money has a price slot. An if-added in the price slot will then find the budget slot in the birthday-party frame and tell its budget frame the name of the frame that just ran the if-added. Then a program is run that checks to see if the budget limit has been exceeded. If so, the budget items and their cost are presented to the user, along with some possible courses of action. A selected item can be deleted, in which case the system merely erases that item from the birthday party frame tree. Another possible action is modification of the choosen frame. This is not always possible. For example, the crummy munchies have a fixed price, $20.00. Some items, like ice-cream, can alter their price by altering quantity. The last course, is to just ignore the price limit altogether.

RELATION TO OTHER WORK

Marvin Minsky

Marvin Minsky's original frames paper [1975] provided much of the original direction in this paper. His original ideas of what a frame should be were vague, but the structure he desired was one that conveniently represented a "stereotyped situation." Along with information telling how to use the frame, what to expect next, and what to do if these expectations do not occur. The frames that are used in this paper know a small amount about how various frames are used (through such things as what kind of frame they are the kinds of slots that they have, etc.). We don't do very much about expectations. We would claim that the information about the plan structure is what is really wanted in order to make the standard sort of shifts from one frame to another.

Minsky's original application for frames, was to represent visual scenes. For example,

a frame might represent a view of a block. But an important feature was not only that the frame captured the relationships between the faces, both visible and invisible, but that each frame also "understood" its relationship to other viewpoint frames. Thus given a set of external relationships or conditions, such as moving the viewpoint to the left, the frame would know what new frame applied to the new context. Minsky's intent was that this same mechanism would carry over into the conceptual realm, where externals, like spacial transformation were replaced by externals like temporal or causal succession.

Minsky claimed that one of the advantages to frames, is that such a mechanism could solve reference problems like a birthday party where Jane intends to buy Jack a kite, but her friend says, "He already has a kite. He will make you take it back." Minsky claims that the key to understanding what "it" refers to, is recognizing that "taking it back" represents a possible thing to happen next, and should be present in the gift frame. Then the problem of linking these actions is merely one of recognizing which slot of the frame is being refered to. This is precisely the position of this paper, except that instead of this being an expectation, this is really a plan branch. We are representing a birthday party as a set of interconnected frames, one of whose major organizational principles is determining plan choices, relationships, and parts. In this case, discourse is merely providing the external conditions that are responsible for selecting one branch of the hiearchy of action.

Eugene Charniak

Charniak [1972] also highlighted many of the problems in knowledge representation. His main technique, was to represent common sense knowledge as expectations in the form of demons. The idea was that a particular semantic concept activated a collection of demons. Each of these demons was expecting a particular consequence from its semantic concept. In a sense this is a frame structure with these demons representing the plan branches. Perhaps this is the aspect of behavior that Minsky was thinking about when he said that frames included expectation, and what to do about them. This more direct link between the semantic concepts and their consequences was not fully realized. Instead, demons worked through a sort of indirect action. Particular actions would change some state in the data base, which would then fire the demons. The explicit relation between succesive semantic concepts was sometimes clouded by requiring the links to be made through intervening states. One example, due to Minsky, is that of returning a birthday present, mentioned before. Charniak's solution was to postulate that Jack did not like the present, this being made a state in the data base, which caused a demon to be fired that expected the gift to be returned. In fact, Jack might love to have two kites, but would just prefer have something else. It seems to be more just the conventional thing to do, return a

present if the recipient already has one. No doubt, feelings and emotions may be involved in the choice of one branch in a plan over another, but the issue is that there seems to be only a reasonably small number of things to do at a given junction. That is the question that people always ask, "what are the options?" The issue that is present with this frames approach, is recognizing when when a move is being made to a particular slot.

Roger Shank

Shank [1975] has ideas about how his individual semantic concepts are formed together to represent a typical sequence of events, viz. scripts. In fact, he states that scripts describe "non-planful" behavior. Rather they are predetermined sequences of actions that define a situation. They have entering conditions (how you know that you are in one), reasons (why you got into one), and crucial conceptualizations. They are associated as the definition of a situational noun (such as a birthday party). They are represented as sequences of conceptual dependencies that represent the temporal ordering of ACTS. There are also extra actions that are conditionally inserted in the sequence, which he loosly refers to as "what-ifs". We presume that by "non-planful" behavior, he means that these scripts are to be taken more in the vein of predicting future actions, based on the specific conditions (through the "what-ifs"). What he refers to as plans are definite sequences of subroutines that are to be handled as units.

Shank uses different terminology, but his views are not too different. His focus is just predicting a temporal sequence, possibly modified according to world conditions. He is not interested in embedding scripts in a structural hierarchy. These are perhaps just artifacts of his using these in language understanding, where it is more important to know about the temporal ordering.

Earl Sacerdoti

Sacerdoti [1975] takes a much different viewpoint than the rest of the people discussed so far. He is very much concerned about the transition from a plan description, much like our birthday party description, to a temporal plan. His focus is one of ordering the plan parts efficiently, and in a non-interfering way. He uses the concept of procedural net. His idea of a node in a procedural net, is quite like our notion of a frame, although his nodes only represent actions. According to him, planning consists of succesively expanding the nodes level by level, critizing the plan at each level. The expansion is very much like the idea of frame instantiation in our system. A particular plan node is expanded into its immediate constituent actions, with their respective links. The main purpose of his critics is

to constrain the possible temporal orderings, and eliminate some redundancies that might arise. When we instantiate a new frame to fill in a slot, there are no critics as such. There are requirements, that may for example require one activity to occur after another. If all of the requirements that the user wants are in the frame, the frame system can inform the user when there is a problem. We do not attempt to provide an optimal solution to scheduling difficulties as does Sacerdoti.

Our basic organization is so similar that his techniques could be used to resolve conflicts resulting from a poor choice of temporal ordering. In fact, a good planning service should offer the user the option of scheduling things for him. Sacerdoti's system outputs the temporal ordering not as a necessarily linear string, but as a net that indicates what things do not strictly have to follow one another. This information would then tell the user what options he had with regard to his scheduling, or could be used to present the user with a sequence, based on the user's preferences. Sacerdoti only attempts the problem of temporal ordering. He still does not help us with the problem of living within a budget. Usually the biggest problem that we have found, is not that there is some sort of difficult time sequencing problem, but that there may not be enough time to everything, that is limited utilities.

CONCLUSION

Using this frames system, many thing could be easily done, but the implementation of several desirable tasks requires a more mature system. Certainly it is straight forward to be able to attach special data handling procedures to every slot. It was easy to tell what should be a frame, but not always so easy to decide how to organize the overall frame structure. Different versions of the birthday party system tried different organizations of frames to try to reflect what was really going on. Sometimes, the system was changed every time that a new aspect of the data was seen. One of the most useful aspects of the frame system, was that data, and such things as requirement criteria, and actually any sort of information that affected the performance of a given slot, could be put out in the open where it was easy to access and evaluate. This makes it easy for programs to look at what is going on and make simple decisions about how to satisfy and possibly change some of these criteria. Too often data is unnecessarily placed in programs in such a way that a very sophisticated understanding of the language, and of programming techniques is needed to decide what the desired effect of the data is. Even so, I suspect that some more careful thought has to be given to the whole concept of putting critical program performance criteria in a more

lucid format. In some sense, there is an analytical problem here of deciding just what the critical points are, and how to represent them. One of the first approaches to this problem was the idea of subproceduralizing each point and controlling them independently through their inputs. This was certainly part of the problem, but people have also felt the need to annotate these. Just putting a subprocedure there does not automatically explain its use. The need is for clarity in representing what really happens with this function with these inputs. The slot idea in frames helps a lot. Because now, not only are the actual proceedures defined, and their inputs defined, but the input data is also in effect commented. That is the data's uses are specified.

It seems that there might also be some extensions of the system that could add desirable features, and make it behave a little more in line with the sorts of goals presented in the introduction. We noted earlier how the price function works. It would be quite simple to extend this feature to other kinds of limited resources, such as time. A more difficult problem is what to do about utilities in general. There are very poorly understood trade-offs between utility (e.g. usefulness, novelty, tradition, etc.) and cost (in money, time, privacy, pride, etc.). It is one thing to calculate these, but quite another to perform bargaining among entities possesing these in various quantities. Several years ago, people learned that the tradition value of Thanksgiving turkey was enough to overcome great monetary cost. The present presents the user with the list of budget items and lets him choose which to change. A system that properly understood these utilities might be able to present real suggestions. There is also the possibility of developing a system that understood better what sorts of things form a good sub-unit for budgeting. In the present system, any frame can set up a cost limit that would in effect not allow the total cost of that frame and the ones below it to exceed the limit. Now all of this is left up to the users discretion, with no advice or suggestions at all.

There have been several other items that would make for a more responsive sort of birthday party system. The existing frames system should be modified to be able to follow a more flexible mode of frame instantiation and filling, as suggested. Some more attention must be given to frame organization and documentation. It is not clear exactly how frames should be organized on a larger scale. As far as mechanism goes, it is certainly true that one can just keep on linking frames into other frames, but there seems to be aspects of overall frame organization that must be made clear. This is, I suppose, why one does exercises like designing a birthday party system. The things that others have called plans and scripts, etc. are not very well understood. In the introduction, there was some hand-waving about subjective and objective criteria for organizing frames. This needs to be made much more explicit.

BIBLIOGRAPHY

[Charniak, E. 1972]
"Toward a Model of Children's Story Comprehension", Technical Report 266, MIT Artificial Intelligence Laboratory, Dec. 1972.

[Goldstein, I. P. and Roberts, R. B. 1977]
"Nudge, a Knowledge-based Scheduling Program", AI Memo 405, MIT Artificial Intelligence Laboratory, 1977.

[Miller, M. and Goldstein, I. P. 1976]
"Overview of a Linguistic Theory of Design", AI Memo 383, MIT Artificial Intelligence Laboratory, 1976.

[Minsky, M. 1975]
"A Framework for Representing Knowledge", in P. H. Winston (ed.), *The Psychology of Computer Vision*, New York, McGraw-Hill, 1975.

[Sacerdoti, E. D. 1975]
"The Nonlinear Nature of Plans", Advance Papers of the Fourth International Joint Conference on Artificial Intelligence, 1975

[Shank, R. C. 1975]
"The Structure of Episodes in Memory", in Bobrow, D. G. and Collins, A. (eds.) *Representation and Understanding.* New York: Academic Press, 1975

[Sussman, G. J. 1975]
A Computational Model of Skill Acquisition. New York, American Elsevier, 1975.