To LRG

From Trygve Reenskaug

Subject A note on DynaBook requirements

Filed on [IVY]<Reenskaug>

Date 22 March 1979

All DynaBook users must have a good understanding of the functioning of their personal system in order to utilize it properly. Most users would like to go further: To build their own variations on top of whatever they have purchased. The claim of this note is to experiment with some structuring techniques and user-oricnted descriptions of Dynabook systems.

Some of the thoughts and ideas behind the note have been given to me individuals in the great community that is called LRG; some are derived from Prokon, a system developed at the Central Institute for Industrial Research in Oslo; and some are the results of my experiments with Smalltalk. I hope I shall never be required to sort out what's what.

The note has been written with the aid of Bob Flegal and Diana Merry's DocumentEditor, which I feel to be an exciting part of the future. Even though its slowness is still maddening, Bob and others are steadily improving it and I recommend it for anybody witl1in LRG to try out.

1. ABOUT THE SYSTEM REQUIREMENTS (p. 2)
1.1 A FUNDAMENTAL REQUIREMENT (p. 2)
    1.2 THE STRUTURING OF THE OTHER REQUIREMENTS (p. 2)
    1.3 ON THE BASIC METAPHORS (p. 3)
2. BUILDING A PLANNING SYSTEM - A SCENARIO (p. 4)
    2.1 BACKGROUND (p. 4)
        2.1.1 Network models (p. 5)
        2.1.2 Other views (p. 6)
    2,2 TOP-LEVEL GOAL (p. 7)
        2.3 TOP-LEVEL SOLUTION (p. 7)
2.4 EXPLORING A POSSIBLE SOLUTION THROUGH A SCENARIO (p. 8)
        2.4.1 The Scenario (p. 11)
3. ON THE STRUCTURING OF OBJECTS WITH MUILTIPLE ROLES (p. 10)
---------------------------------- Remainder is not scanned ---------------------.-
4. BASIC METAPHORS (p.)
    4.1 TOP-DOWN DESIGN AND IMPLEMENTATION (p.)
    4.1.1 What we need to describe (p.)
        4.1.2 Typification (p.)
        4.1.3 Hierarchy (p.)
        4.1,4 The programming system (p.)
    4.2 BOTTOM-UP DESIGN AND IMPLEMENTATION (p.)
        4.2.1 What we need to describe (p.)
        4.2.2 Virtual computers (p.)
        4.2.3 Programming (p.)
    4.3 THE PROGRAM EDITOR (p.)
APPENIDIX: A PLANNING SCENARIO (p.)
    1 The First Model (p. 22)
    2 A Composite Network Model (p.)
    3 Separation of Model and View (p.)

1. ABOUT THE SYSTEM REQUIREMENTS

1.1 A FUNDAMENTAL REQUIREMENT

The international standardisation organisation (ISO) has published a standard vocabulary for data processing. To me, the definitions of data and information are very important, saying profound things about the nature of data processing. As I recall them from memory, they run as follows:

Data. Formalized representation of facts or concepts in a form suitable for transmission, interpretation or processing by humans or automatic means.

Information. (in data processing) The meaning a human assigns to data on the basis of an agreed convention.

A data processing system is only capable of handling data. These data must be interpreted in the system's environment. In the design of traditional systems, a large amount of work is spent in selecting a powerful set of abstractions, deciding on suitable representations for these abstractions, and ensuring that all users attach substantially the same meaning to them. Once a system is installed, there is a large amount of inertia in the total man-machine system which tends to cement old decisions whether they were good or bad.

This should never be the situation for DynaBook. The user community for a given DynaBook is a single person. The abstractions needed by this person will not be known, neither to the person nor to anybody else. Further, the needs for abstractions will change with time. I suggest we add two new articles to the declaration of human rights: *Man should not be required to know his needs,* and *Man should be permitted to change his mind at any time without being penalized with slave labor at his keyboard*. The consequence of this is the following simple requirement:

It must be trivial for a user to modify his DynaBook system at any time both in terms of representations and abstractions. Such modifications apply equally to the programs and the current data base.

I regard this as a fundamental requirement, overriding all other requirements.

1.2 THE STRUCTURING OF THE OTHER REQUIREMENTS

The other features of DynaBook could be specified in many different ways. From a data processing point of view, we could for example use the headings:

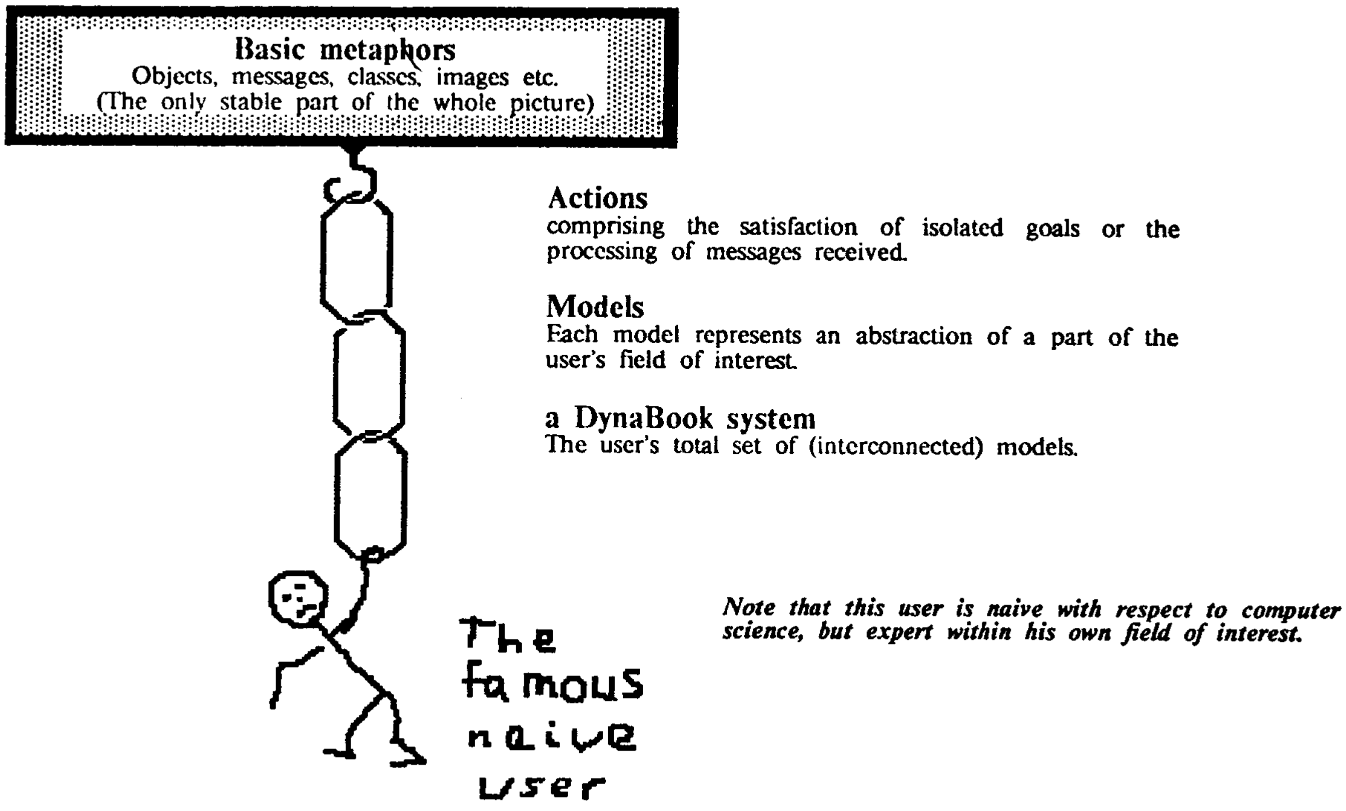
Manipulation of data. Computations and other transformations.

Storage and retrieval of data. The information retrieval function.

Communication of data. The movement of data from one place to another.

All data processing systems provide all theses services, but for different purposes and with varying success. DynaBook must, of course, provide them in a simple and unified way.

To my mind, a more user-oriented organisation of requirements is would be beneficial. I am giving one possible list of headings below. It is presented as a chain, because I regard the weakest link to be the most important at any given point in time.


Basic metaphors
Objects, messages, classes, images etc.
(The only stable part of the whole picture)
Actions
comprising the satisfaction of isolated goals or the processing of messages received.
Models
Each model represents an abstraction of a part of the user's field of interest.
a DynaBook system
The user's total set of (interconnected) models.
Note that this user is naive with respect to computer science, but expert within his own field of interest.
The famous naive user

Maybe it is too ambitious to try to cover all three levels at once. But by 1982-86 there are bound to be a lot of hardware products on the market which look like a DynaBook. They are likely to be programmable in APL, an extended Basic or some dialect of Pascal; and should provide excellent facilities for supporting individual actions. But they are not likely to support the user in all of her varied endeavours through models and total systems. Such support could be one of the main selling points of DynaBook.

1.3 ON THE BASIC METAPHORS

In keeping with the above considerations. let us try to find out something about what the user will need in terms of basic metaphors.

As I see it, the typical user will want her DynaBook do something that initially is only represented as a vague feeling. She then needs a language (with a set of basic abstractions) for thinking about the new problem and for mapping her thoughts into her DynaBook system. In many, if not most, cases. the new model will somehow be linked to older models with attached data that already exist in her DynaBook.

Most problems would start with a rather unclear and often self-contradictory goal. Some examples: I would like to get better control over my finances; I don't want to be troubled with detailed accounting; I want to settle my account with the butcher; I want to know more about Tarot cards; or I want a small. cheap house with many large, luxurious rooms.

I would expect the user to go more or less subconsciously through a goal-means hierarchy: Certain means are needed to reach a given goal. These means are not immediately available. but constitute a new set of part-goals, each of which needs certain means for their satisfaction, and so on.

An example: To get better control over my finances, I would need to set up a budget; to keep account of all income and expenditure; and to keep a running comparison between budget and accounts. Three new, non-trivial goals that need further consideration.

At some point. we should end up with a set of consistent and well-defined goals. Ideally. the DynaBook should take over from there, and solve the problem automatically.

Unfortunately. it seems inconceivable that a standard DynaBook should contain the solution to all possible problems. The user must therefore be prepared not only to state her problem. which is difficult enough. but also to describe the method for its solution.

We therefore need some metaphors that are closer to the computer. We need something that can help us in splitting the solution into smaller parts and to specify how each of these parts is to be carried out. If the same metaphors could help us in describing not only tIle solution. but also the problem. this would be near perfect.

Presently, the only real contenders at this level are the object/message metaphors. I also think that they may provide the basis for a very satisfactory solution.

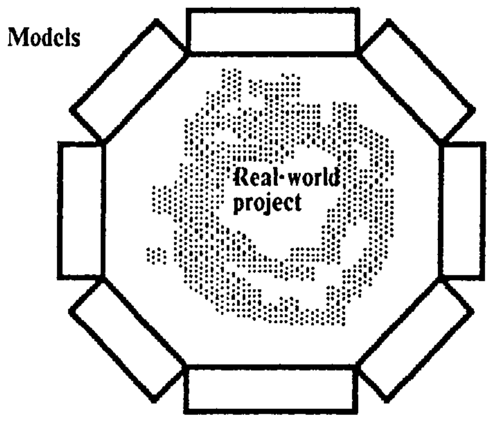
Before going into the details of this. it may be useftll to sketch out one possible description of a fairly complex system. The sample I have chosen is based upon the planning system that Dave Robson and I are working on.

2 BUILDING A PLANNING SYSTEM - A SCENARIO

2.1 BACKGROUND

Controlling large projects is a complex task with a very large number of interdependent details. Those responsible for the project have to keep track of all these dependencies in order to understand the consequences of various real or proposed situations that may arise.

A number of manual and computer-assisted systems are used as aids in this task. Examples are systems for the establishment of the requirements for materials; purchasing systems; store control systems; various rough, preliminary planning systems; time registration systems; cost estimating and control systems; and various budgeting and accounting systems.



I prefer to think of all these systems as implementations of different models of that elusive something which is the project itself.

The models are represented somewhere within the total office system as a collection of data together with the procedures necessary to process these data.

The models will usually be overlapping in the sense that several models may have their own representation of the same real-world fact. The interdependencies between the models are taken care of through various means of communication between the models.

Ideally, all models should be totally consistent. This ideal is not attainable in practice because it would require an overwhelming bureaucracy and stifling rigidity. One should therefore accept some inconsistencies, the aim being that the total set of models should be a reasonably accurate representation of the project without spending too much effort on trivialities.

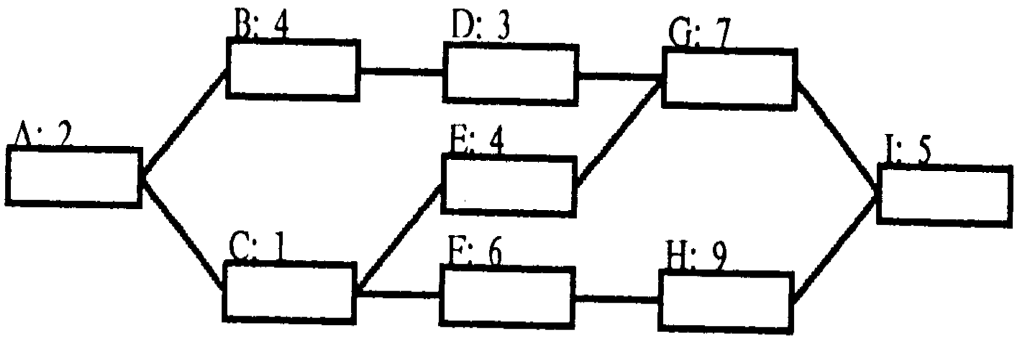
The network model is often regarded as the single most important model in the project because it links together what should be done with who should do it and when it is to be done. We therefore focus our attention on network models, keeping in mind that there must be room for a multitude of other models in the overall framework.

2.1.1 Network Models

In this type of model, every task that has to be performed in the project is mapped into a simple element called an activity. Basically, an activity is characterised by its duration and its predecessors. The duration specifies the time needed to perform the corresponding task, and the predecessors are the activities that have to be completed before the present activity can be started.

Based upon the activities' predecessors, it is easy to find the successors of each activity. The start activities of the network are all activities that have no predecessors, and the end activities are the activities that have no successors.

In the figure below, activities are represented by rectangles, dependencies (successor-predecessor pairs) by lines, and the duration of each activity is printed above its rectangle.



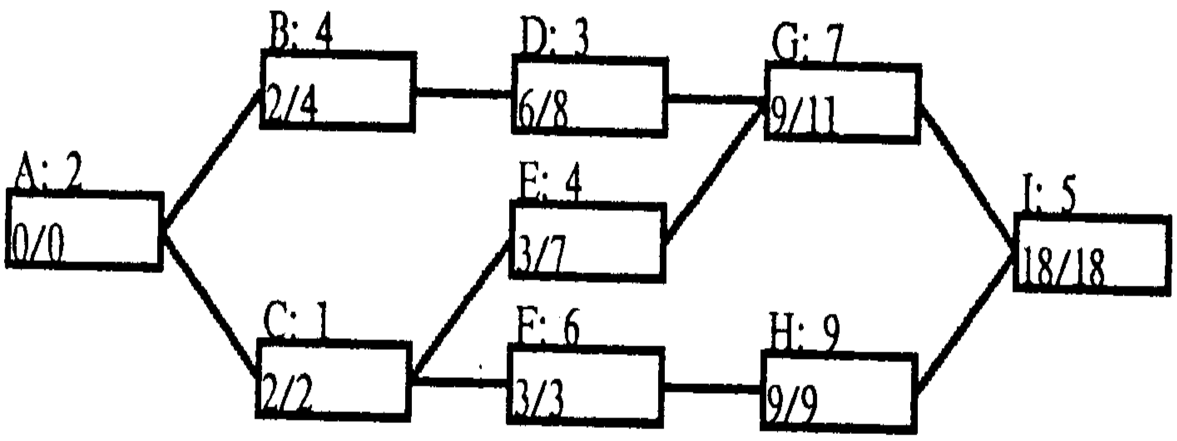
In this model, a dcpendcncy always means that an activity may not sun before its predecessor is completed, Other types of dcpcndencics are also needed in practice: A successor may start some time arter the start of its predecessor, an activity cannot be completed before its predecessor is completcd, and so on. Funher, actual delays may be given spccifying a minimum time delay bctween the neighbouring activities. The above model is therefore oversimplified. but it is sufficicnt for our prcscnt discussion.

If the start time for the whole project is known, this time is, per definition, the early start time for all startactivities. By following the successor-chains and using the duration of each activity, the early start time may be computed for each activity. The late finish time for each activity is simply its early start + its duration. These calculations are called *frontloading*.

The latest of the early finish times for all endactivities is the earliest possible completion time for the whole project. With this as a starting point, the late start and late finish times for all activities may be calculated. These calculations are called *backloading*.

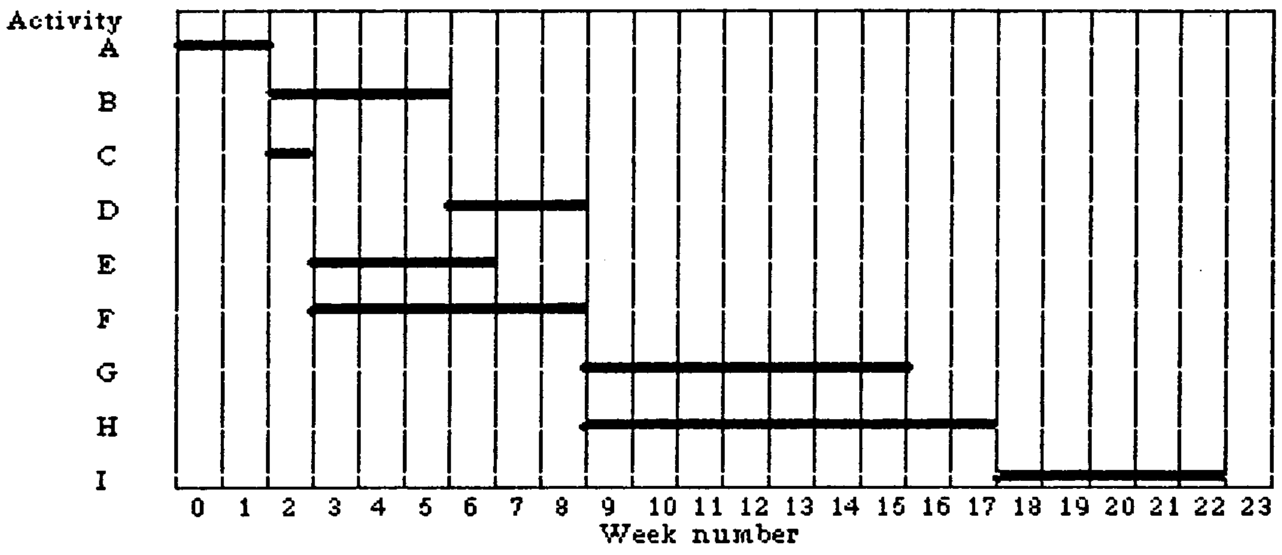
The difference between the early and late start times for an activity is called its *float*, it expresses how much the performance of that activity may float in time without endangering the completion time of the project as a whole. The smaller the float, the more important it is to follow the progress of that activity carefully.

The early/late start times for each activity in our example are shown in the diagram below.

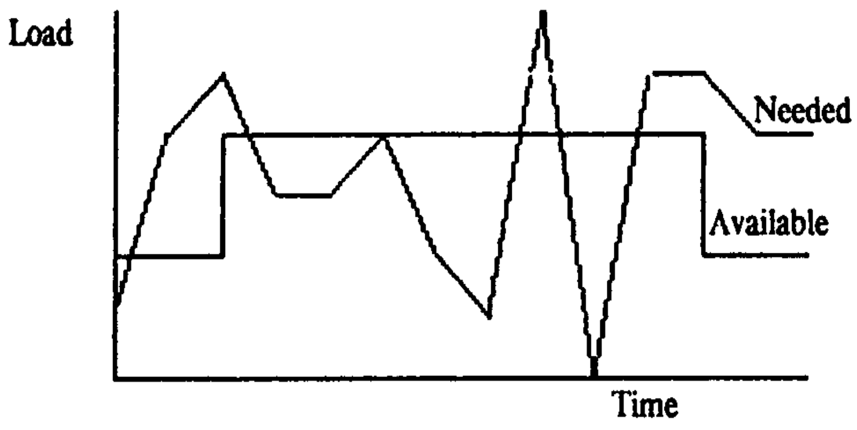


2.1.2 Other Views

A realistic schedule for our Simple project could be made by deciding to start all activities as early as possible. This plan could be presented in a variety of forms. Different tables are commonly used. Another form is the Gantt diagram as shown below.



Most activities need some resources in order to be carried out. The resource requirements may be summed up for each resource category, and compared to the total amount of resources available of that category:



When the plan is first made, the requirements may show large variations with time. Activity times are then adjusted within the available float to even out the load on critical resources.

Resource requirements are also commonly presented in the form of tables. This gives higher accuracy and makes it more difficult to get an overall impression of the situation.

Many activities imply monetary expenditure, and client's payments may be connected to certain milestones in the project. Cash flow information is therefore often included in the network model. Adding such extraneous information into the network model makes it more complex and tends to camouflage the basic simplicity of the mode1 concept. Further, not all expensces can be associated with activities in a natural manner. I therefore prefer to keep the models 'clean', and to concentrate the accounting information in the accounting models.

2.2 TOP-LEVEL GOAL

Our first observation is that our imaginary client's control of large projects is not satisfactory. Activities are all too often behind schedule, important decisions and events are not always reported to all that should know about them, the consequences of delays are not always fully understood so that necessary action may be taken in time.

Our second observation is that all competent personnell at every level in the project organisation are highly stressed. The sheer size of the job makes its successful completion an almost impossible task. All hands must concentrate on the primary goal of the project: To get the project finished on time and within budget. Project control bureaucracy is regularly experienced as being of secondary importance and seems pointless from the viewpoint of the individual.

Our top-level goal is thus simple and easy to understand:

We want to provide a tool that reduces the workload imposed on the project staff for the purposes of project control. At the same time, we want a substantial improvement in the quality of the same project control.

2.3 TOP-LEVEL SOLUTION

The second part of the goal seems to be best served through improved project control procedures, more detailed follow-up of plans, better and more accurate reporting. In short, more formalism and more bureaucracy.

The first part of the goal seems to be best served if we could place more trust in the individual member of the project staff so that there would be less need for detailed reporting. and follow-up. The project control overhead could then be reduced, resulting in less formalism and less bureaucracy.

The first solution appears to be in direct opposition to the second one, and we may have to make a choice. Since the second solution most certainly will put an even higgher stress on the project organisation, we will concentrate our efforts on increasing individual responsibility, reducing formal procedures and finely tune the procedures that are left in the system.

Two very important conditions must be fulfilled if this solution is to be a success. First, the organizational climate must be such that the individual is encouraged to take responsibility. This requires a conscious cultivation of the best aspects of the present organisation and perhaps some innovation in management principles. Second, each individual must be provided with the background and information which that particular person needs to exercise the increased responsibility.

The first condition is primarily a human problem, and must be approached as such. The satisfaction of the second condition is a technical problem, however. We feel that it will be necessary to provide each person with a flexible, on-line tool that permits him or her to select the data that is of current interest from all datcl available in the project; and to browse through, compare and manipulate these data within a unified framework.

For the total system to work, it will need accurate and timely reports from each person working in the project. Whenever possible, such reports should not have to be generated as isolated items, but should fall out as a by-product of the user's main tasks at his work-station.

The aim of all tool development should be to provide each individual user with the best possible tool for that user. The tools must be highly individualized, both as regards the user's function in the project and his personal work habits and preferences.

Highly individualized tools do not necessarily entail large amounts of special software development for each individual. Hardware and general purpose software is likely to be provided by the manufacturer. Professionals within the user's fields of interest would build specialized software modules on top of that general system. Computer professionals within the user's organisation would select a subset of such modules, and provide the general framework for the organis.1tion's information processing. Finally, the individualization will consist of the high-level selection, adaption and integration of the modules into the user's set of computer tools. This last individualiz.ation should ideally be done by the user himself, but could also be done by professional assistants.

There are two critical questions that need an answer:

Is it possible to provide sufficiently practical and powerful tools at a reasonable cost?

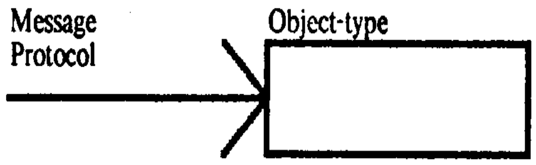
Is it possible to make tl1e systems sufficiently flexible and easy to understand so that it becomes feasible to individualize them?

An experiment within the framework of Smalltalk will hopefully give some first, tentative answers to these questions. Further, since our problem is to device a tool for building individualized project control tools, we are not interested in engineering yet another project control system. We will rather try to let such a system evolve gradually, always basing our next step in the development on our experience with the previous one.

2.4 EXPLORING A POSSIBLE SOLUTION THROUGH A SCENARIO

Dave Robson and I have been doing a series of experiments with project control systems using Smalltalk- 76. In this scenario, I try to sketch out a similar set of experiments, this time based upon a modified Smalltalk-x. The purpose of this is to explore possible changes to the basic Smalltalk concepts and programming aids.

In Smalltalk-x, a protocol is a named collection of message patterns and their meanings. An object-type is a typical representative of a collection of objects that all recognize the same protocol(s). All objects belonging to the same object-type do not necessarily share the same programs.

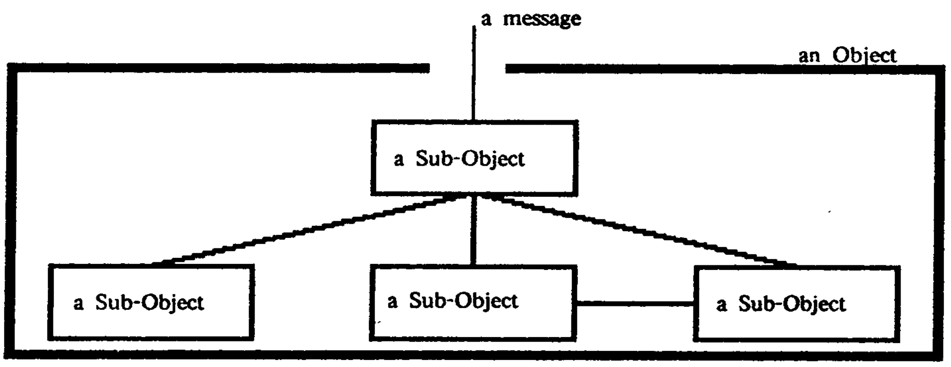


Object-type
Fields:
name(String-type) *"description"*
...
Protocol
message pattern "description"

An object will remember its state in state variables. Such variables (fields) properly belong to the internal description of the object rather than the external. They will be shown together with the external message protocols, however, because they define potential message paths and because they are needed in the discussions on how to modify an active Smalltalk-x system. A field is typed through describing the protocol(s) of all objects it may point to.

When an object receives a message, an action takes place within the object. As seen from the outside of the object, an action is observed as an indivisible operation.

When an object is 'opened up' to see how an action is actually performed within the object. we usually find that the object consists of a number of interacting sub-objects. All the object's actions are implemented as an organized interaction between these sub-objects.



A *prototype* contains one particular implementation of an object-type. There are no superclasses, but a prototype may inherit characteristics from one or more other prototypes.

2.4.1 The scenario

I envisage a DynaBook user who wants to develop a program for the drafting of networks on his screen. He does not have much programming experience, but he is an expert in project control. He naturally starts off with the simplest possible program, and uses this program to capture data about his project as these data becomes available, thus building up a valuable data base in his DynaBook.

As time goes on, the shortcomings of this first program becomes intolerable. The user then modifies the program, taking care that the data already collected are not lost in the process.

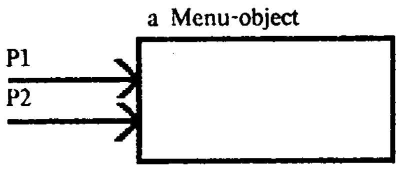
In the scenario, I am focussing attention on the message protocols. These protocols may be defined cxplicitely as shown, but they may also be defined implicitely through the definition of methods.

The scenario itself is enclosed as an appendix to this note. Two conclusions may be drawn. They are (1) that a typical system is likely to evolve along a path of least resistance rather than by careful design and implementation. And (2) that an evolving system often implies an evolving data base, since there typically will be valuable data in the DynaBook that is mapped onto the current version of the system.

3 ON THE STRUCTURING OF OBJECTS WITH MULTIPLE ROLES

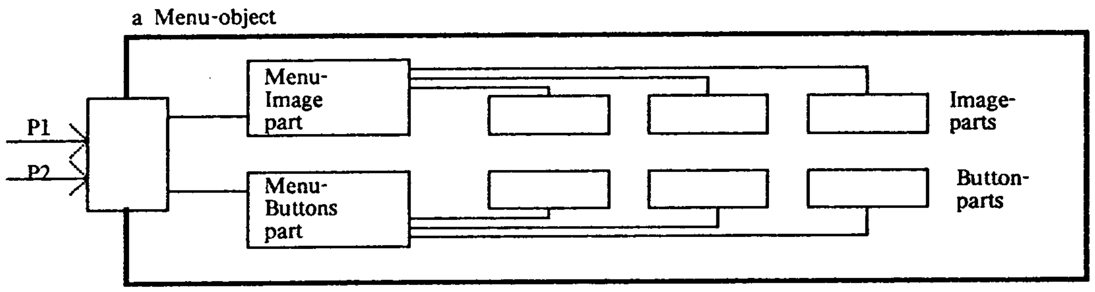
The function of an object may often be divided into a number of separate roles; this may help to increase standardization and the sharing of code. A simple example is the title field to the left on the upper edge of a window: it is a rectangle, a paragraph that may be edited, and a title.

A more complex example is a Menu with a number of Buttons that the user may bug. The external view of a Menu could for example be as shown below.



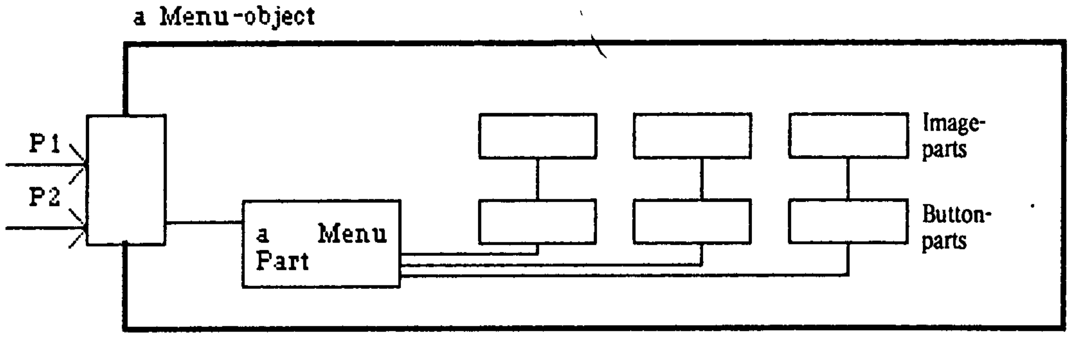
Here, PI is the protocol for defining the appearance of the menu on the screen, for positioning it and showing it. P2 is the protocol for finding out if the user has bugged one of the buttons, and to get hold of the identifying string of the button that is pushed.

Protocol PI could be very well served with the protocols of an image, while protocol P2 would likely be special for menus. One internal representation of a menu.object could therefore be as follows:

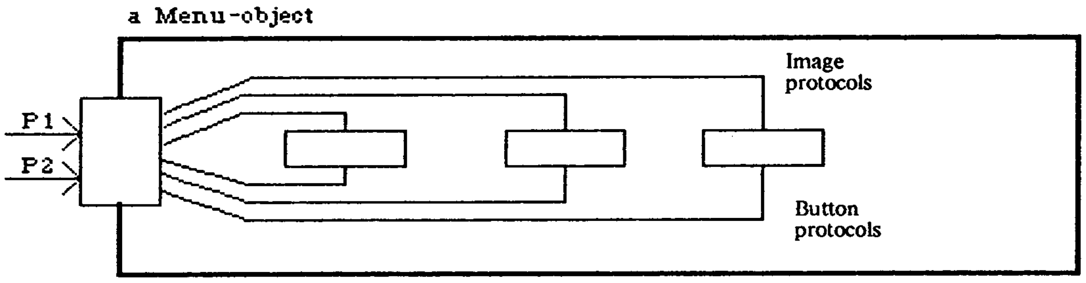


In this solution, there are two sub-objects for each button in the menu: a general Image-part and a more specialized Button-part. All the Image-parts are controlled by a super-image (Menu-Image-part) and all the Button-parts by an object Menu-Button-part. The solution does not appeal to me, because I would like to regard a Button as something having both an appearance on the screen and the characteristics of a menu item. Further, I would have to store the Button rectangle in two places: In the Image for showing it on the screen, and in the Button for finding out if the user has bugged that button.

A somewhat more appealing solution would be to let each Button-part own its Image:



The only excuse I can see for this asymmetric solution, would be that it allowed us to use some existing code. I would much prefer to represent each button with all its attributes as one object. If we insist on being able to inherit characteristics from code that already exists (and we do that), a Button-object should have inherited some characteristics from Images and some from Button-parts. The structure would be sometl1ing like this:



4 BASIC METAPHORS

The remaining pages from original 14 to 32 have not been scanned.