Some Contrasts and Considerations of an Approach to Modelling

Martin S. Feather
USC/Information Sciences Institute
4676 Admiralty Way
Marina del Rey, California 90291

1. Introduction

We consider the approach to modelling that our group has been developing, first to highlight differences between this and the approaches of other researchers, second to raise some issues related to understandability of models that we feel are common to modelling in general. The principles underlying our approach may be found in Balzer's position paper to this workshop.

2. Differences

Underlying many of the differences is our decision as to what should and should not be part of the model. We suppress both implementation and interface details from model of the functional behaviour (intending that these details be separately stated). This allows us to concentrate on issues fundamental to the functional behaviour of the task. Our models are operational in nature, hence may be executed in order to exhibit the described behaviour. We forgo the ability to have our models automatically compiled into efficient implementations in order to permit use of constructs best suited to behavioural specification. We will examine several of these constructs and comment on the extent and limitations of them.

This research was supported by RADC contract F30602-79-C-0042 and DARPA contract DAHC 15 72 C 0308.

* The views expressed are those of the author.

2.1. Information storage and retrieval

We model information by relations among (typed) objects. Information retrieval may follow these relations in any direction (e.g. given a 3-place relation R, and objects A and B of the appropriate type, we may ask for an object C which could fill the third position R(A,B,C), or given B and C seek A, etc.). We may quantify over all objects of a given type (e.g. ask for all objects C which are related to A and B in R(A,B,C), or ask if there exists any such object, etc.). We are limited, however, to relations among objects (we may not have relations among relations) and quantification over objects (so we may not ask for all relations in which an object participates), i.e. our language is first order. In practice these limitations do not seem to restrict our ability to model tasks.

2.2. Modelling of change and use of historical reference

We model change by the creation / destruction of objects and the insertion / deletion of relations amongst them. Each such primitive operation causes a transition from one state to another - thus our computation history is a series of states. All information in any previous state is available through "historical reference". Indeed, state is treated as a first-class type, i.e. we may quantify over states, instantiate an object of type state (e.g. to ask for a state in which ...), etc. (But some operations make no sense applied to states, we may not destroy or change states, since history is inviolate.)

In addition to historical reference, which is a convenient way of extracting information about the past without having to explicitly remember information from state to state, we also have a limited form of "future reference"; we may perform an action in the current state, extract information from the resulting state, and continue, with the information, from the state *prior* to starting that action.

Non-determinism and constraints

Non-determinism enters our model through the selection of an object (e.g., some x such that...) or through the sequencing of operations (e.g., forall x in <set> do <action> denotes an arbitrary sequencing of applications of the action to the elements of the set). This is reflected in our state transitions by the possibility of a state having multiple alternative successors.

The constraints we may impose on our model are expressed as arbitrarily complex predicates, which may involve any information of the current or earlier states (the latter through historical reference). Any transition which leads to a state in violation of any constraint is anomalous. In the event of all transitions from a state being anomalous, the transition that led into that state is by definition also anomalous. Thus constraints serve to "prune out" undesireable paths. This provides a mechanism for formally specifying desired behviour without specifying an "algorithm" for achieving it.

2.3. Limitations

We pointed out the first-order limitation in what information may be stored within our model; another limitation concerns the use of data-typing. First, we do not permit user-defined parameterised types. For convenience the language provides sets and sequences as parameterised type constructors, but these are the only such instances. Second, we do not follow the abstract data type approach of associating operations with types. This divergence from the trend of modern programming languages is a consequence of our very different approach to modelling. Our aim is to model as directly as possible the structure and behaviour of the task domain and activities, and our constructs are designed to let us do this. In contrast, we view models constructed by building up layers of abstraction from more basic types and operations as an indirect description of the structure and behaviour through implementation in terms of the basic types and operations.

3. Issues relevant to understanding

We have constructed models of two real-world examples using our specification language: a "Source Data Maintenance" package to support batch-commands from a user maintaining source code arranged into a hierarchical structure of units, files, libraries and projects; and part of the ARPA-net Host-Imp protocol, involving issues of parallelism, modelling hardware failure, etc. We base our observations upon our experience with these (and consideration of other) models.

We have found that readers of our models often find it hard to understand them - even other members of our group who have equal fluency in the language. This is somewhat disturbing since we had hoped that by making it easier for the specifier to more directly model the task domain, reading and understanding of such models would become easier too. Although we could no doubt derive some improvements by making cosmetic changes to our language (the development of which has been biased so far towards easing the task of the constructor rather than the reader), we feel there are fundamental issues in presenting models which deserve more attention, and which apply to all approaches to modelling, not just our own. We consider why models are hard to understand, and what might be done to alleviate this difficulty.

3.1. Sources of difficulty in model understanding

For any sizable and complex task, a model of the same is likely to be sizable and complex. The reader, faced with the task of comprehending the whole of a large model, typically finds no assistance to help him build up gradually to an understanding of the whole.

The design decisions taken by the model creator will be apparent only through the form of the end product. As such, the reader will find little or no trace of the reasoning the constuctor followed.

The interactions of the various portions of the model, and their implications for the behaviour they describe, may be unclear to the reader. The viewpoint of the constructor may be different from the viewpoint of the reader, and unsuitable to answer the questions the user might have.

3.2. Aids to understanding

We suggest the following tools might be of assistance in understanding models:

A symbolic interpreter would let the reader study the behaviour of the model on classes of (rather than particular) examples. Since our approach to modelling is based upon an operational model of the world, we can build such an interpreter for our language. For this and other tools it is important that the user be able to guide the application of the tool and its presentation of results. For an interpreter we might want to involve the user at non-deterministic choice points, so that he can direct the interpreter along the paths which most interest him. The user should have means for stating which portions of the behaviour he wishes to observe during the interpretation, and be able to examine the resulting state.

Analysis tools could be used to point out possible interactions between different portions of the model. The more sophisticated the tool the better able it would be to identify true sources of interactions and discard non-arising ones. One particularly appropriate method for such tools to present results might be to generate small examples of interactive behaviour.

A tool to produce a switch of viewpoint (e.g. following the progress of an object through the model rather than observing individual processes) would free the user from the need to adopt the constructor's viewpoint.

These tools act upon the finished model to assist understanding. In addition, we might seek to make the development of the model a manipulable object in its own right. If the model is developed in layers of increasing complexity, the reader may later follow through the development in order to incrementally attain an understanding of the whole. Again, a tool would be of assistance, this time to help the constructor perform his development of the model and record his progress. This possibility complements the earlier suggestions - the user might apply the other tools to investigate the behaviour of an intermediate layer of the model. Finally, the existence of the development as an object may assist maintenance of the model - if some change was to be made, the change could be made at the appropriate layer of model, and development adjusted from that point.