Computer Science 773

Robotics and Real-time Control

PROGRAMME DESIGN

THE IDEA.

This method was devised by an avionics firm as a way of producing a reliable specification of software for complex systems. The important goal is to ensure that nothing is missed out, and the method therefore puts great emphasis on determining that sources for all data required for each step are defined. This method records, but does not explicitly, check time constraints; the information it gathers identifies the various types of process which are involved in the system, and defines the quantities required by a method such as rate-monotonic scheduling.

The method is organised as a programme for compiling the complete specification document. (They call it a requirements document; terminology varies from expert to expert.) The compiling process is completely predefined, following the argument that if you deal with each case individually you're likely to miss out steps which don't obviously apply, and thereby perhaps miss information which you should have. It's better to have a lot of items marked "not applicable" because you *know* they're not applicable than to risk missing a few pieces of information which later turn out to be essential.

Several principles of procedure are laid down in the source paper. One such principle governs the conduct of the interviews which might be necessary in seeking the required information :

> *Decide (write down) what you're going to ask before you ask it.*

If you don't, you are likely to ask only the questions with easy answers, and particularly those which the respondent believes that you ought to be asking. If the answer isn't immediately available, don't go on to any later stage which might depend upon it until it has been supplied. That's another principle :

> *Don't start things before you have all the prerequisites.*

If you do, you are likely to waste time repeating steps which you should only have to do once – or, worse, you won't notice that they need repeating, and you end up with a bad specification.

On the basis of these and other principles, the standard programme for completing the specification document is drawn up. It's based on a set of forms which they insist should be filled in; they believe that the formality is important, and leads to better results than you can get informally by making notes from conversations. They think it works.

THE DETAILS.

The process begins with the table of contents, which is always the same :

Notice that the headings cover a wide range of information. (It's interesting that there's no place marked to say what the system does !)

Thereafter, much of the process is reduced to filling in standard forms. The forms for specifying the software functions are of most interest to us; some specimens follow.

The first step is to define the input and output requirements. The input data item form is used to describe every source of input. The example describes a switch, giving information about what it does, what it looks like to the operator ("switch nomenclature"), and so on.

The output data item form does the corresponding job for each output value : it says what it's for, where it goes, what it looks like, and so on.

With the input and output settled, the functions can be defined. The procedure is to work backwards from the output, at each step defining a function which calculates on of the undefined items. Notice that a classical top-down analysis is difficult, because there are many potential "tops", all of which might interact. This parallel method of defining a solution is found to be effective.

There are two sorts of form for defining calculations; these are called the demand function and periodic function forms. The distinction is based on when the function is required; a demand function is required when some identified event, or one of a set of identified events, occurs, while a periodic function is required at regular time intervals. (Recall the process types handled by the rate-monotonic algorithm.) Each function controls the value of exactly one variable, which might be of a compound type.

The demand function form is presented as a decision table. There is a column for every possible action of the function (that is, every assignment of a value to the associated variable), and a row for every relevant system state. The tabulated values are the events at which the actions must be taken.

The periodic function form is similar, but in this case the union of the conditions in any row must be "always". (Work it out.) The bottom row in the example is the action, though it's not labelled as such; compare the label with the name of the function.

As a final example, here is a rather less well defined item – the "undesired event" list. This is a collection of everything that can go wrong, and is not easy to summarise in a simple form. Instead, they give what amounts to an algorithm for reviewing the items defined in the other parts of the exercise and finding out what could go wrong in each case. Here is some text and a table to describe their approach :

COMMENTS.

The method is strongly based on a functional decomposition of the process. Not everyone thinks that's a good idea, but it seems to fit real-time applications rather well, because it's the functions which correspond to the procedures which have to be scheduled when the system runs. The resulting system has a pronounced object-oriented flavour, with a function, perhaps containing several methods, defined for each variable.

Alternatively, one could define it in terms of decision tables, or as a rule-based system; if nothing else, this illustrates how similar many of these ideas are.

REFERENCE.

K.H. Britton : "*Specifying software requirements for complex systems*" (Proceedings of IEEE conference on specifications of reliable software, 1979), as reprinted in R.L. Glass : "*Real-time Software"* (Prentice-Hall, 1983).

Alan Creak,
April, 1998.