Making Specification Design More Accountable

Dave Robertson, Jane Hesketh
Department of Artificial Intelligence,
University of Edinburgh,
80 South Bridge, Edinburgh
Email: dr@uk.ac.ed.aisb, jane@uk.ac.ed.aisb

Abstract

If something goes wrong with a software/hardware implementation then it is important to know what caused the problem. Sometimes the error lies in incorrect implementation of a specification but it is thought that a more common source of problems lies in earlier stages of the development process, where there may be mismatches between the requirements of the domain and the specification. For this reason, it is common (particularly in safety–critical applications) to find tightly monitored regimes of specification, in which the strategies of specification designers follow prescribed conventions and are closely constrained by regulatory reviewers. It would be useful if this process were as explicit as possible, making the design and reviewing process more open and accountable. One way of doing this is to provide formal descriptions of design strategies and to require designers to endorse their use of these by reference to appropriately formalised parts of the regulations. We examine how this may be done, using as a concrete example the domain of oil platform emergency shutdown systems.

1 Introduction

Our general approach is to follow an existing methodological framework, using formal methods to reinforce key parts of existing practice. Figure 1 gives a sketch f the main components of this framework and their interactions. Our prime concern is to provide an acceptable system specification, which we would expect implementation designers to realise in appropriate software and/or hardware. To be acceptable, the specification must address more than just the behavioural requirements stipulated for the domain of application. It must also be considered "well designed", by conforming to standard practice for the the specification language concerned. Furthermore, it must have been "carefully designed" – that is, designers should be able to show that they have taken appropriate codes of practice for the domain into account when constructing the specification. Ideally, they should be able to annotate appropriate parts of the specification with evidence to show that they have paid attention to standard design principles and codes of practice. This is what we mean, in this paper, by accountability in specification design.

To ground our discussion in a realistic domain, we consider the use of this methodology in specifying oil platform emergency shutdown systems. Oil production platforms are complex and expensive systems in which it is necessary to place humans in close proximity to potentially hazardous industrial processes. This situation is sometimes caricatured as "building a hotel next to a chemical plant". It is therefore essential that the emergency shutdown systems on production platforms function reliably and that the design of such systems is performed in a highly disciplined and accountable manner. We shall begin our discussion by presenting an account of the specification methodology of this domain (following the template given in Figure 1). We next describe a mechanism for formalising part of standard design practice (using schematic design components) and then show how this can provide a framework for attaching endorsements on design decisions in terms of parts of a code of practice. We conclude by raising some questions which we believe are particularly germane to this line of research.

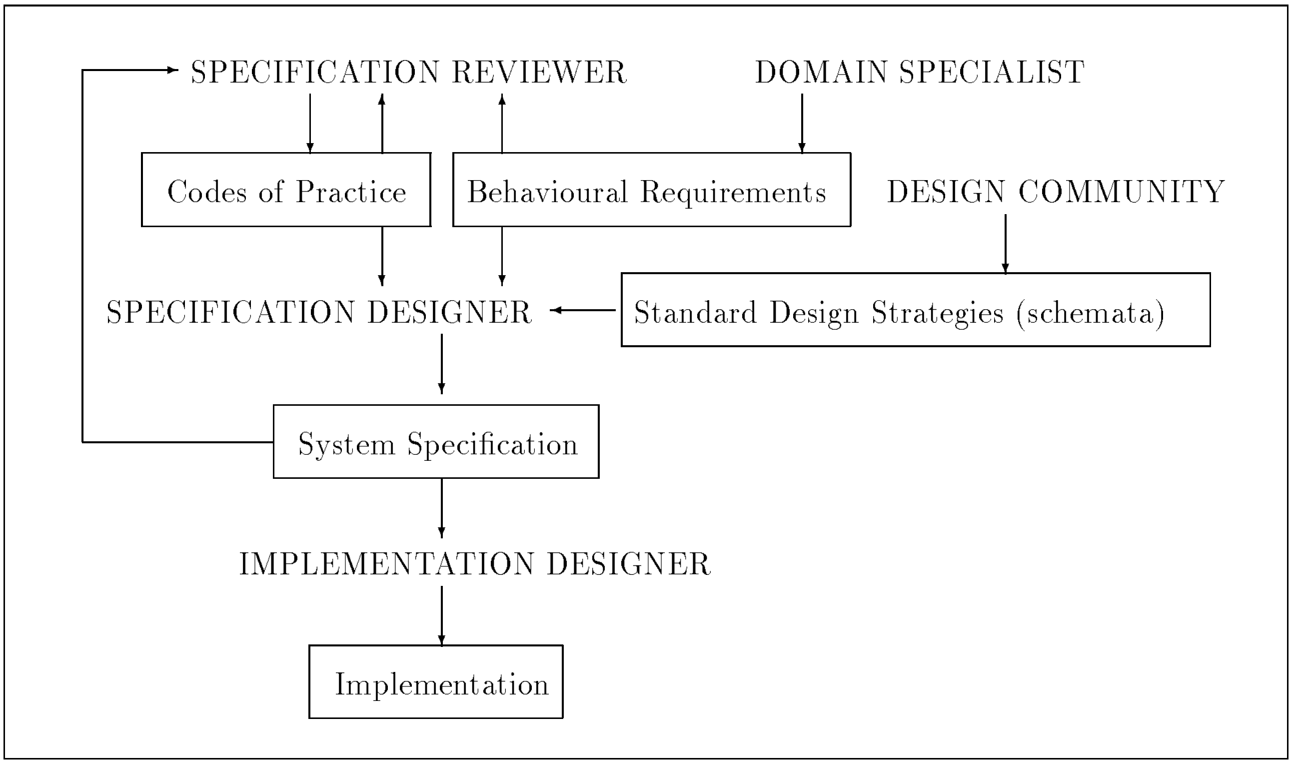
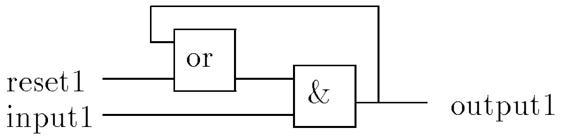


Figure 1: Schematic diagram of one type of specification process

2 Sketch of the Shutdown System Design Process

The entire process of production platform design and commissioning is highly complex – involving collaboration between government, oil companies and numerous subcontractors. We shall concentrate on the core of this process, which involves the requirements stipulated in an oil company's code of practice and its interaction with the specifications used as the starting point for system design. Because of the high risks involved, the company provides a detailed code of practice for emergency shutdown and process trip systems. This contains a mixture of information, ranging from general goals which must be achieved by the system engineer (*e.g.* that all the relations stipulated in the cause–effect matrix should be accounted for) to specific requirements for system behaviour (*e.g.* that trips should latch outputs in a de–energised state). The design of a shutdown system begins by determining the inputs to the safety system (from devices such as sensors or trip switches) and the outputs (which activate alarms, shut down machines, etc). A typical safety system might have 1500 inputs and roughly the same number of outputs. To provide a coarse–grain description of the relationships between inputs and outputs, a cause–effect matrix is drawn up. This gives, for every input cause, the output effects which should be initiated if the input is tripped. Tripping is normally signalled by de–energising, thus ensuring that power failure initiates safety shutdown. Given a cause–effect matrix, the next step is to design a specification for the safety system such that the stipulated cause–effect relations are obtained and that conforms to the company code of practice requirements. The standard language for specification is "functional logic", which is a diagrammatic language based on standard logic circuit elements with time delay components. There is a high degree of standardisation in the construction of these specifications, with similar configurations frequently being re–used.

Strong links between requirements and specification are important at two stages in the evolution of the specification. The most obvious of these is in representing standard patterns of design and in



$value(output1, Time, Value) \leftarrow$	% A $Value$ for $output1$ at $Time$ is obtained if
$\quad previous\_time(Time, Tp)$ &	% Tp is a the time point before the given $Time$ and
$\quad value(output1, Tp, V0)$ &	% $V0$ is a value for $output1$ at Tp and
$\quad value(reset1, Tp, V1)$ &	% $V1$ is a value for $reset1$ at Tp and
$\quad value(input1, Tp, V2)$ &	% $V2$ is a value for $input1$ at Tp and
$\quad or(V0, V1, Vor)$ &	% Vor is the logical 'or' of $V0$ and $V1$
$\quad and(Vor, V2, Value)$	% $Value$ is the logical 'and' of Vor and $V2$.
$value(output1, Time, Value) \leftarrow$	% A $Value$ for $output1$ at $Time$ is obtained if
$\quad initial\_value(output1, Time, Value)$	% $Value$ is the given initial value for $Output$, with
	% $Time$ being the initial time.

Figure 2: Reset configuration: diagram and FOPC interpretation

linking these to the appropriate details of the code of practice during program construction. This provides a formal framework for specification, making the connection between design decisions and company requirements more explicit. A less obvious, but perhaps more important, role is during later maintenance of the system. If part of the implemented system needs to be changed then it is important to ensure that this alteration remains consistent with the code of practice. However, the connection to the code of practice from the specification diagrams may not be obvious, unless the original designer has kept detailed records of the links between the two. In the next section we shall describe the techniques which we have employed to maintain these types of links.

3 Specification Schemata

In Section 2 we noted that designers frequently re–use variants on standard configurations of specification components. We refer to such configurations as schemata. One of the simplest examples of a schema is the reset configuration, an instance of which is shown in Figure 2. The diagram shows the standard method of displaying this configuration with an annotated Horn clause interpretation of this diagram appearing below it[1]. A reset configuration is used to allow a trip to be reactivated after deactivation. Thus, in the circuit of Figure 2, if $input1$ is deactivated then $output1$ should be deactivated and remain so until $reset1$ is activated at the same time as $input1$. Note that the Horn clause interpretation we have given for the functional logic is not necessarily the "correct" one, since this depends on the level of detail with which one wishes to model the behaviour of the circuit. We shall return, briefly, to this issue in Section 5.

If we examine the circuit in Figure 2, we can see that it contains elements which distinguish it as a reset component: a feedback loop from the output to a logical 'or' with the reset, with the resulting value being 'and'ed with the input. However, there is flexibility in this definition – in particular, we could cater for a conjunction of more than one input signal simply by installing a sequence of 'and' connectors (one for each additional input, with the output from each 'and' being carried forward to

[1]We adopt the Prolog convention that constants are represented with words beginning in lower case letters, whilst variables begin with upper case letters and are implicitly universally quantified

$$
\begin{aligned}
& schema(reset, Agenda, Output, \\
& \quad \{\ value(Output, Time, Value) \leftarrow \\
& \qquad\qquad previous\_time(Time, Tp)\ \& \\
& \qquad\qquad value(Output, Tp, V0)\ \& \\
& \qquad\qquad value(Reset, Tp, V1)\ \& \\
& \qquad\qquad value(Input, Tp, V2)\ \& \\
& \qquad\qquad or(V0, V1, Vor)\ \& \\
& \qquad\qquad AndGoals, \\
& \quad\ \ value(Output, Time, Value) \leftarrow \\
& \qquad\qquad initial\_value(Output, Time, Value)\}, \\
& \quad reset\_cluster(Reset, Inputs), \\
& \quad generate\_and\_sequence(Inputs, Tp, Vor, Value, AndGoals), \\
& \quad \{Reset\} \cup Agenda).
\end{aligned}
$$

Figure 3: Reset configuration schema

the next). Thus, we can define a general reset schema which, when given the appropriate subset of inputs which need to be allocated a reset, will generate the necessary circuit specification. In general, our schemata have the form:

$schema(Name, Agenda, Output, Axioms, Problem, Constructor, Agenda')$

where: $Name$ is the name of the schema; $Agenda$ is the set of outputs which need to be accounted for in the design at the time of applying the schema; $Output$ is the output which the schema produces; $Axioms$ is the set of axioms necessary to produce the output; $Problem$ is the design requirement which needs to be satisfied in order to use the schema; $Constructor$ creates the necessary axiom structure; and $Agenda'$ is the set of outputs which need to be accounted for after applying the schema. An example of a general schema for the reset assembly described above appears in Figure 3. The design requirement for this schema ($reset\_cluster(Reset, Inputs)$) is that in the requirements there should be some combination of $Inputs$ which have to be connected with the given $Reset$ switch. The $generate\_and\_sequence$ constructor produces the appropriate sequence of 'and' connectors, thus instantiating the variable $AndGoals$.

4 Constrained Generation from Schemata

The schemata of Section 3 can be harnessed to provide a goal–directed mechanism for generating functional logic specifications, starting from the outputs of the cause–effect matrix and working "backwards" to account for each of these in terms of the inputs. The essence of the agenda–based generation algorithm is described in Figure 4. Note that the search space for this algorithm is potentially infinite, since the application of a schema can increase the size of the agenda of outputs. The key to the successful operation of this algorithm is to be highly selective in the choice of schema for each output in the agenda, so that the specification is not only as compact as possible but also remains faithful to the requirements of the code of practice.

In Figure 3, we have already given an example of a requirement at the level of the design itself which helps to constrain the selection of appropriate schemata. This was the schema problem condition: $reset\_cluster(Reset, Inputs)$, which collects the necessary reset and input identifiers for a reset configuration. We expect the decision about which these should be to be taken by the specification designer (assisted as far as possible by our system). However, the designer must also indicate why

Assume we are given a cause–effect matrix with set, $\mathcal{I}$, of inputs and set, $\mathcal{O}$, of outputs. The predicate $generate(\mathcal{I}, \mathcal{O}, \{\}, \mathcal{A}_F)$ will, if successful, produce a set $\mathcal{A}_F$, of axioms describing the functional logic specification linking $\mathcal{I}$ to $\mathcal{O}$. This is defined recursively as follows:
$generate(\mathcal{I}, \mathcal{O}, \mathcal{A}, \mathcal{A}_F)$ when:

- If $\mathcal{O} = \{\}$ then $\mathcal{A}_{\mathcal{F}} = \mathcal{A}$.
- Otherwise, remove an element, O_1, from $\mathcal{O}$ leaving remaining elements $\mathcal{O}_R$ and:
  - If O_1 is already accounted for in $\mathcal{A}$ then $generate(\mathcal{I}, \mathcal{O}_R, \mathcal{A}, \mathcal{A}_F)$.
  - Otherwise, select a schema such that:
    * $schema(Name, \mathcal{I}, \mathcal{O}_R, Axioms, Problem, Constructor, \mathcal{O}_N)$
    * and the conditions in $Problem$ can be confirmed from user requirements.
    * and $Constructor$ can generate the necessary structures in $Axioms$.
    * $\mathcal{A}_N$ is the result of merging $Axioms$ with $\mathcal{A}$.
    * and $generate(\mathcal{I}, \mathcal{O}_N, \mathcal{A}_N, \mathcal{A}_F)$.

Figure 4: Simplified schema application algorithm

this choice was made by referring to the code of practice, otherwise it will be difficult to see how these more general requirements influenced the design. An example of a relevant general requirement from the code of practice is:

> "Attempted Operation of any reset facility with a trip demand still active shall have no effect."

This requirement does not, directly, determine the details of the specification design – it imposes a condition on the behaviour of the completed system. Thus, the choice of a particular instantiation of the reset schema of Figure 3 might be endorsed by appealing to the above excerpt from the code of practice. By forcing designers to make their assumptions explicit in this way, it becomes easier for other experts to assess whether they are warranted.

Where requirements in the code of practice are sufficiently detailed, we may provide formal interpretations of these. However, since sentences in natural language typically admit a variety of interpretations the way in which we formalise requirements at this level may not be uniquely determined. In other words, there may be more than one way of demonstrating that a requirement holds. Taking the reset requirement above as an example, one interpretation of this might be that if there is an initiating signal to one of the trip devices then there should be no combination of inputs to the other devices which can reactivate one of the outputs which it was designed to trip. Figure 5 gives one way of formalising this requirement. The axiom in this figure says that if I is a trip device and A is an action which should follow from I (as indicated by the cause–effect matrix and the initiatin signal for I is V_I and $\mathcal{S}$ is the set of all inputs other than I and the activation signal for A is V_A, then it should not be possible to find an time later than some initial time point, T, such that an assignment of activation values can be made to $\mathcal{S}$ and, using these, the activation value V_A is derived for A. This more detailed rendering of the natural language requirement could be used by the designer to stipulate the type of verification which he/she recommends as a way of confirming that the requirement has been met.

Given the functional logic axiom set, $\mathcal{A}$:

$$
\begin{array}{lll}
\multicolumn{3}{l}{trip\_device(I)\ \&\ action\_of(I,A)\ \&} \\
\multicolumn{3}{l}{initiation\_signal(I,V_I)\ \&\ activation\_signal(A,V_A)\ \&} \\
\multicolumn{3}{l}{\mathcal{S} = \{X | (input(X)\ \&\ \neg X = I)\}\ \&} \\
initial\_time(T) & \rightarrow & \neg(\ assign\_values(\mathcal{S},T,\mathcal{V})\ \& \\
 & & later\_time(T,T_F)\ \& \\
 & & \{input\_value(I,AnyT,V_I)\} \cup \mathcal{V} \cup \mathcal{A}\ \vdash\ output\_value(A,T_F,V_A))
\end{array}
$$

Figure 5: Requirement endorsement

5 Conclusions

We have described, in simplified form, a system for constructing specifications of shutdown systems using schemata to capture standard design strategies for portions of the system, with a goal–directed specification generator tying the portions together. Our generation algorithm if unconstrained, would be capable of generating a huge variety of possible configurations so there needs to be some way of constraining the search. This is done by associating preconditions with each schema, determining its applicability to the problem which the designer is tackling. Designers must also endorse their choice of schemata by appealing to appropriate excerpts from the code of practice. Some of these excerpts may be interpreted formally – thus allowing designers to stipulate what they consider to be appropriate tests of compliance with the requirements.

The domain chosen for our experiments is, we feel, particularly appropriate for the use of formal methods. The high stakes involved in oil production, plus the strong emphasis on accountability and monitoring, mean that the effort of producing a formal framework for interpreting requirements may be justified. However, we have applied a similar approach, with some success, in the domain of ecological modelling [2]. In the ecological domain, the emphasis was on the use of requirements as a way of buffering users from the details of specifications of ecological models. By contrast, in the domain of safety shutdown systems we expect users to be highly conversant with specification details, with requirements being used primarily as a means of endorsing design decisions.

The strongest limitation on our system is the range of design strategies encompassed by the library of schemata which we provide to designers. Our approach works best when there is a high degree of standardisation in specification style. If users wish to stray beyond these limits then we must provide them with general purpose schemata which either require a high degree of parameterisation by the user or individually contribute only small portions of the design. One way of achieving generalisation in a more disciplined way, in the context of Horn-clause specifications is to use notions of specification "techniques", enabling incremental construction of predicate definitions. A summary of recent Edinburgh work in this area is given in [1].

A further, fundamental, issue in the formalisation of requirements in systems such as ours is the degree of expressive power which we allow users in describing their problems. Highly expressive languages, although perhaps theoretically appropriate, may not easily be understood by users in the domain. Simpler languages may be easier to present to users but may provide insufficient information to control the generation of specifications. In [3] we have examined this issue in the context of our work in the ecological modelling domain.

References

[1] A. W. Bowles, D. Robertson, W. W. Vasconcelos, M. Vargas-Vera, and D. Bental. Applying Prolog Programming Techniques. *International Journal of Human-Computer Studies*, 41(3):329–350, September 1994. Also as Research Paper 641, Dept of Artificial Intelligence, University of Edinburgh.

[2] D. Robertson, A. Bundy, R. Muetzelfeldt, M. Haggith, and M Uschold. *Eco-Logic: Logic-Based Approaches to Ecological Modelling.* MIT Press (Logic Programming Series), 1991. ISBN 0-262-18143-6.

[3] D. Robertson, M. Uschold, A. Bundy, and R. Muetzelfeldt. The ECO program construction system: Ways of increasing its representational power and their effects on the user interface. *International Journal of Man Machine Studies*, 31:1–26, 1988.