Experience of using a lightweight formal specification method for a commercial embedded system product line*

Michael Breen

Abstract

A simple specification method is introduced and the results of its application to a series of projects in Philips are reported. The method is principally designed to ensure that every unusual scenario is considered in a systematic way. In practice, this has led to high-quality specifications and accelerated product development. While the straightforward tabular notation used has proved readily understandable to non-technical personnel, it is also a formal method, producing a model of system behaviour as a finite state machine. In this respect, the notation is unusual in being designed to preserve as far as possible a view of the overall system state and how this changes. The notation also features a constraint table which may be described as a kind of spreadsheet for invariants to help define the states of the system.

lightweight formal methods, finite state machines, tabular notations, embedded systems, user interface specification

1 Introduction

For some time, there has been an increased recognition that for formal methods to be adopted widely in industry, they must be lightweight [1]: that is, specifiers should be able to enjoy formal method benefits such as lack of ambiguity and amenability to automated checks but without needing to learn arcane mathematical notations or subtle theoretical concepts. A successful method must also be more than merely a notation; it must support requirements specification as an activity. Thus, Potts emphasizes the significance of systematic methods for requirements elicitation over the mere invention of languages to represent them [2], and Parnas observes that, "The industry badly needs methods that will help inspectors to proceed systematically, carefully considering all cases in a way that provides confidence that nothing has been overlooked" [3].

Here, I describe a method which was first developed to specify the externally-visible behaviour of a CD recorder and which is designed primarily to support this kind of systematic elicitation. Further advantages include a notation that is easily understandable to the general reader and which supports a naturally incremental style of requirements definition leading to the creation of a formal model of system behaviour as a finite state machine (FSM).

The model produced by the method differs from most FSM models in its decomposition and in its limited use of structure. While much progress in software engineering has come from finding new structuring principles and mechanisms, types of structure vary according to their effectiveness in particular cases and are optimized for different purposes. In the context of a requirements specification in particular, there is some justification for considering reduced or alternative kinds of structure and the process by which a structured representation is arrived at. For example, it has been observed that the use of object-oriented methods can lead to information which domain experts would keep together being distributed throughout a specification [4, 5]. Also, it may be difficult to visualize a sequence of interactions for a user interface when it has been specified as a set of general rules, any of which may apply at any time [6]. Experimentally, subjects have found it more difficult to read specifications in which modularizing information led to lots of page-flipping; further, even though a hierarchically-structured specification was judged easier to read than an equivalent flat specification, the subjects had unknowingly made errors in reading it which they did not make with the latter [7].

A significant distinction of the approach introduced here is that it structures a finite state model of behaviour

*Requirements Engineering Journal 10 (2).

not as a network of single-variable FSMs but rather as rules which show the change in state of many variables at once. The tabular format used for this, called a transition table, is a refined version of one that has been suggested before [8], however its application to a substantial industrial project has not previously been reported. It should be emphasized that this decompositional approach differs not only from that of graphical notations but also other FSM-based tabular notations such as SCR [9]: it disposes of all communication between individual single-variable FSMs, effectively replacing them with a multi-variable FSM. This organization means that one does not need to work through a sequence of internal events or a dependency network to see the overall response of the system in a particular case. Crucially, it means that one can take a systematic approach to reviewing the overall behaviour rather than merely that of the component FSMs considered separately.

This alternative approach is obviously limited in scalability. Later, I will outline how this was addressed in practice through a compromise with conventional structuring approaches. However, even in its simplest form, it has been possible to apply the method to a non-trivial system and to gain significant benefits from it.

As the method was originally developed to address the exigencies of a particular project, it is convenient to begin this article with the background to that project and to explain how the method was designed to address the problems that arose. Following this is an account of the application of the method to the original system and subsequently to related systems. I then outline and contrast comparable existing techniques before concluding with a summary of the main results.

2 Industrial Background

The project which motivated the method and to which it was first applied was the Philips CDR870, which was to become the first audio separate compact disc recorder (CDR) aimed at the consumer end of the market. The key factor for the project was the short time available for development: it was regarded as commercially essential that the product be on the market by the following Christmas. Meeting the schedule imposed by this deadline and various lead times would define success or failure.

The critical component for determining schedule feasibility was the application-level software, which was largely responsible for the behaviour of the system as perceived at the user interface. Unlike the lower-level software, this was to be developed from scratch and the consensus among those who had worked previously on systems of similar complexity was that this was not possible within the time available: about six months from specification to integration.

Two people, including the author, representing a Philips design house, felt that it was possible to develop the software within the schedule. However, this would depend crucially on having a very high quality specification with almost no omissions or other defects. This level of quality was needed because we could not afford the cost in time of resolving issues discovered later in the project lifecycle. Additionally, our estimates depended on the preparation of a suite of automated regression tests in parallel with the application code development, with the personnel for this task working largely independently, from the specification.

Specification of the user-visible behaviour therefore began on a provisional basis, a final decision being deferred until progress on this would allow a more reliable assessment of feasibility. Because of the urgency of the schedule, it was agreed to proceed more-or-less straight to the construction of a formal model of the external behaviour, in co-operation with the person responsible for deciding on the details of the user interface: as a finite state machine model, the specification would be precise and unambiguous, thus helping to fulfill our quality requirement; it would also be essentially executable, thus helping to reduce implementation effort.

The specification began well. For example, it was easy to model the behaviour of the CDR's tray as a finite state machine which responded to the pressing of the open/close button by changing from the "closed" state to the "opening" state. However, for this example, there is an exception: if the system happens to be recording when the open/close button is pressed then the convention is to treat this as a mistake and ignore it (forcing the user to press the stop button first). And, as more functions were incorporated into the model, the number of unusual scenarios to be considered increased substantially, with many of them being easy to miss. Not unusually for a system with a user interface, the complexity of the CDR was therefore quite disproportionate to its small size. Thus, it was possible to consider the behaviour of individual components, like the tray, and satisfy oneself that they were correct, only later to discover cases in which the overall system response was incorrect.

The main problem was the lack of any systematic way to find these unusual scenarios, or corner cases. Initially, we believed that diligent effort would reduce the number of such cases that had been overlooked to the very low level necessary. As specification progressed, however, there was diminishing confidence on the part of the author in particular, as principal modeler and "scribe", that this point could be arrived at quickly enough. Further, it would be difficult to know that we had reached that level of quality – or when we were likely to reach it – with the degree of certainty one would wish in committing to a project with a hard deadline and minimal margin for scheduling error.

Some other problems might in part be ascribed to the particular notation used for the model, which was Statecharts [10]. It was known from the outset that the person deciding the details of the user interface would have difficulty reviewing the statecharts. However, we understood there would also be difficulties with other notations based on the concept of communicating finite state machines. Given the pressing circumstances, it was agreed that he would sign off on a specification in this form trusting that it accurately modeled the behaviour discussed in the meetings. Nevertheless, this was obviously not ideal for either party and negated some of the advantages of having a formal specification.

Since states may be organized hierarchically in Statecharts, the notation also permits a relatively high degree of structuring. However, this was not an advantage during the specification process as it was difficult to make good structuring decisions while the behaviour was still being elicited. For example, a decision to nest certain other states within the "tray closed" state might initially seem elegant but later prove awkward when trying to incorporate other elements into the model. Weighing such structuring decisions involved unwanted effort that would be better deferred to a separate design activity; changing any such decision additionally carried the risk of introducing errors into the specification by failing to preserve some correctly-modeled behaviour.

The ideal specification we sought would therefore:

- require the minimum of structuring consistent with avoiding inefficiency, thus making it easy to record the system behaviour incrementally as it was established;
- be easy to understand without compromising on the advantages of a formal, model-based description;
- above all, admit of a systematic approach so that we could be sure we had overlooked no unusual scenarios.

3 Basics of the Method

The approach eventually used to specify the CDR is now described. This description is organized according to the sections of the original specification and uses modified extracts from that document as examples. The key idea behind the method is this: Rather than constructing a model composed of separate elements interacting as necessary to produce the desired overall behaviour and then simulating this model to verify its response in some sample of cases, instead model the overall behaviour to begin with.

3.1 Variables

The specification begins with a list of the variables defining the overall system state and their possible values, including an explanation of the semantics of any variable or value where these are not self-evident. Figure 1 illustrates.

Figure 1: A CD recorder variable.

Mode

The mode is the *requested* mode. For example, if the play key is pressed when the *Tray* is *open* then *Mode* becomes *play* and *Tray* becomes *closing* – but the CD will not be played until the tray finishes closing.

play

pause

stop

record

3.2 Constraints

The next section is concerned with defining the possible states of the system, that is, which combinations of variable values are possible. For example, if *Tray* is *open* then *Mode* cannot be anything other than *stop*. Such a constraint, or invariant, may be expressed in a constraint

table, a novel notational element shown in Table 1. In the constraint table, the single shaded cell in each row may be thought of as the "if" part of a set of "if - then" statements, with the other cells in the same row giving the "then" parts. Thus, the two cells shown from the first row of the table indicate that if *Tray* is *open* then *Mode* must be *stop*. Similarly, from the row in which *play* is shaded, we read that whenever *Mode* is *play*, *Tray* must be either *closed* (with the CD being played) or *closing* (in order to play it). The symbol '*' means "don't care." For example, if *Tray* is *closed* then the table tells us we cannot infer anything about the value of *Mode*.

Table 1: Constraint table for CD recorder.

Tray	Mode	(other variables . . .)
open	stop	. . .
opening	stop	
closing	play pause stop	
closed	*	
*	stop	
closing closed	play	
closing closed	pause	
closed	record	
. . .		

The shading progresses diagonally down the constraint table with each variable value appearing in exactly one shaded cell. Note that the contents of cells below the diagonal could be derived from those above it or vice versa. However, this redundancy allows easy reference: one may see the effect of a variable having a particular value simply by finding the cell in which that value is shaded and looking across the same row of the table.

Obviously, the format provides for the systematic identification and representation only of simple, or "two-dimensional," constraints, that is, those involving only two variables. In the case of the CDR870, analysis using software developed subsequent to the project helped later to uncover a few applicable constraints that cannot be represented in this way, hereafter called complex constraints. Although none was included in the original specification, a complex constraint may in general be listed separately from the table using a formula, together with an explanation.

3.3 Events

The Events section of the specification simply lists every event to which the system responds, i.e., anything which may cause the system to change state. Figure 2 shows some of the events from the CD recorder specification.

Figure 2: Events for CD recorder.

KEY_OPEN_CLOSE

> The "open/close" button is pressed on the front panel or on the remote control.

EV_TRAY_CLOSED

> The (closing) CD tray reaches the fully closed position.

EV_END_TRACK

> While playing a CD, the end of the current track (song) is reached.

EV_FAST_SEARCH_TIMEOUT

> The "fast" button has been depressed continuously for two seconds.

3.4 Rules

The core of the specification describes the behaviour of the system through a comprehensive set of transition rules. Each rule specifies the response of the system to one of the events for some set of states. The rules are described in a transition table as illustrated by Table 2.

The first rule in this table (with identifier *r1_1*) specifies what happens when the "open/close" button is pressed and *Tray* is either *closed* or *closing* and *Mode* is either *stop*, *pause* or *play*: in all of these cases, *Tray* changes to *opening* and *Mode* changes to *stop*. The following rules illustrate the use of the "no change" ('=') symbol. These appear in the target row of the rule; the row above this is called the source row.

Table 2: Transition table for CD recorder.

Tray	Mode	...
KEY_OPEN_CLOSE r1_1		
closed closing	stop pause play	
opening	stop	
KEY_OPEN_CLOSE r1_2		
open opening	stop	
closing	=	
...		
EV_TRAY_CLOSED r2_1		
closing	*	
closed	=	
...		

Table 3: Rule with non-finite state element.

Tray	Mode	...	Program	...
KEY_DIGIT r7_36				
closed	program		empty entered	
=	=		entered maximum	
The digit pressed could not be the first digit of a two-digit track number on the disc but corresponds to a single digit track number on the disc. It is appended to the program.				

The rules for each event explicitly show the behaviour in every possible state: those cases in which there is no response are recorded using rules with target rows containing only "no change" symbols.

However, not all events are possible in all states, e.g., a "tray closed" event clearly cannot occur if the tray is already closed. Such facts can be expressed using event-related constraints. For the CDR specification, these constraints were self-evident and were omitted. Whether obvious or not, one would typically record such constraints anyway, if only to avoid spurious errors from automated checks; however, no software to perform such checks was available at the time.

Textual notes added to the CDR rules were useful for reminders, explanations, or simply to describe parts of the behaviour which could not be represented as a finite state machine. For example, the note shown in Table 3 describes both a pre-condition and a postcondition with respect to the "program", that is, a sequence of tracks selected by the user for playback (called "memory" on some CD players).

Note also the variable called *Program* in Table 3. This partially models the state of the programmed sequence of tracks using the values *empty*, *entered*, and *maximum* to capture the main equivalence classes, i.e., whether there is a program and if there is whether it has room for any more tracks in it. Abstracting significant non-finite state aspects in this way was designed to give these elements equal prominence and ensure that no unusual cases involving them were overlooked.

Notes added to the CDR rules were also useful in the relatively rare cases where using the tabular format alone for finite state aspects would have led to inefficient, repetitive rules. For instance, for those cases in which the "repeat" setting should toggle, the appropriate table cell indicates that its next value is nondeterministic (like that of the *Program* variable in Table 3); however, the note

Repeat toggles in the following sequence...

restores the missing information in a way that reduces the number of rules required. Although the problem didn't really arise for the CDR, a similar strategy addresses more general cases in which sticking to the conjunctive normal form of the tables would cause a geometric increase in the number of rules. Formulas may also be used rather than natural language.

3.5 Display

The specification ends with a description of the contents of the display on the front panel of the CDR as a function of the system state. The format of this is similar to that of the transition table, but with the target rows replaced by a desciption of the display output. In effect, this is simply a kind of decision table. Table 4 illustrates.

Table 4: Condition table for CDR display.

Tray	Mode	...
open opening	*	
"OPEN"		
closed	play pause	
The track number and the elapsed track time, e.g. "3 1:10".		

4 Results and Discussion

4.1 Initial Results

Completed, the original CDR specification uses 16 state variables, the values of which allow for almost 10^9 system states; however, taking the constraints into account, fewer than 400,000 of these states are possible in the model. The system behaviour is defined in response to 50 different events using a total of 365 transition rules.

The method proved completely successful in eliciting the unusual scenarios that needed to be considered: Every possible combination of event and system state had to be covered by some rule and filling in all of the cells in the tables ensured that no dependency or detail of the response was overlooked. Whenever a new variable was added to the model, a new column of empty cells was created in existing rules, serving as a reminder of the potential dependencies or changes that remained to be considered as a result of the addition.

Specification also proceeded incrementally at a steady pace. The effort of deciding how best to incorporate new elements into the model was disposed of: as each new aspect of the behaviour became known, a new rule was created; completing it simply entailed filling in the table cells for that rule. Because each rule was essentially self-contained, showing one overall system response independent of the other rules, there was never a danger of inadvertently affecting behaviour already modeled.

In terms of process, there were other advantages too. A rule could be added for any event at any time, as could a variable, thus supporting the random way in which information could become available during specification. However, in practice, priority was given to completing the rules for each event in turn: this provided focus in terms of what to do next, meant similar rules tended to be considered around the same time, and made it relatively easy to judge progress and how much remained to be done.

From the customer's viewpoint, the tabular format proved immediately understandable to the user interface expert. Further, at a meeting to decide on whether to proceed with the project, the other main customer representative in the specification process said that he now believed that the project could be done within the schedule – and he identified the specification, although then still incomplete, as the reason for his change of mind.

A specification should be organized as a reference document rather than an introductory narrative about the system: though it is more difficult to browse, it is better in the long run as the information is in a form that allows easy reference throughout the project [11]. In the case of the CDR specification, one could confirm the response for any case simply by turning to the pages for the appropriate event and finding the applicable rule. As a comprehensive and detailed oracle, it reduced the danger of spurious problem reports from integration testers and it forestalled potentially time-consuming arguments on the details of the system behaviour (a particular problem for user interfaces, where everyone has an opinion): not only did it unambiguously state the required response, its explicit form helped to confirm this was not an unforeseen accident of implication; in effect, it declared, "Yes, this specific case was considered and this is what was decided."

The organization of the specification also allowed regression tests to be prepared with a minimum of additional documentation: to ensure broad coverage, tests were associated with all of the rules; each test referenced the rule it was principally designed to exercise through its unique identifier in the specification.

However, in addition to the main specification, it was useful to have an introductory narrative, called the User Requirements Specification (URS). This was a function-oriented document written in natural language by the customer, a little closer in style and organization to a user's manual – however, unlike the specifications described in [12], it was not in fact designed for this purpose and contained technical notes on issues such as ensuring recorded CDs complied with CD standards.[1] For the URS, no particular effort needed to be made to avoid ambiguities or identify exceptions:

[1]For later projects, the equivalent document was called the Customer Requirements Specification.

it was understood that the tabular specification would have precedence in case of any conflict. The two documents served complementary purposes: while the main specification answered every possible "what if" question, anyone unfamiliar with the system would find it much easier to discover the answer to a "how to" question from the URS; besides, a prior knowledge of the system functionality in broad terms was assumed in the tabular specification.

The organization of the specification into independent specific rules proved also to have another minor disadvantage. During the design phase, for which there was a return to the use of Statecharts, it was found difficult in a few instances to model the specified behaviour because it was "inconsistent"– not in the formal or mathematical sense of transition rules contradicting each other, but rather insofar as different responses were sometimes specified in cases which were somehow similar. Apart from being undesirable in itself, this made the behaviour difficult to model in an elegant way: In effect, the approach could be said to have allowed too much flexibility – so that it was hard to generate a specific response using the same synthesis of more general rules that would successfully reproduce the rest of the specified behaviour.

In general, these incongruent transition rules had arisen in cases where more than one possible response was acceptable and, in the absence of a particular preference, one had been selected more-or-less arbitrarily. Change requests were raised for these cases and these were approved immediately by the customer. The problem thus had no material impact on the schedule, with the number of rules concerned also being small. However, it does show how, compared to a model made up of general rules, a model decomposed into specific cases changes the cognitive load on the user: the former emphasizes general rules at the expense of hiding their implications in specific instances; the latter shows the specific cases, but obliges the user to remember guiding principles consistent with existing rules that might be applied to other cases.

The only major difficulty with the CDR specification was checking it manually for consistency and, especially, completeness. Though its form afforded a systematic approach, to ensure that the rules for each event covered all possible cases was in practice a tedious and mentally draining task. This of course was not a difficulty inherent in the method but rather a consequence of having no tool support at the time.

Unsurprisingly, there are several errors in the specification which software now available helps to uncover. First, the constraint table contains a few constraints that are too strong, either becuase of the careless omission of a value from a list of values in a constrained (i.e., unshaded) cell or because an "almost true" constraint does not, in fact, always apply. These mistakes are reasonably obvious once revealed and did not transfer to the implementation. As mentioned earlier, there are also several applicable complex constraints missing from the specification. While the constraint table is not designed to elicit constraints that cannot be represented in it, neither is it essential to the method that all appicable constraints be identified: the worst case consequence of an omitted constraint is that an automated consistency check fails to reveal an error elsewhere, that is, a transition rule which makes reachable a state that should be impossible.

An error of this last kind in a transition rule of the CDR specification may be instructive. In the rule, a variable which is "no change" should actually change to a specific value. Unfortunately, a displayed message obscures any visible effect of this change until a subsequent rule is applied, the effect of which is to progress an operation begun in the first rule. In fact, the modeled behaviour would also be correct if the variable changed in the second rule instead; it is simply that the latter convention would "fit" less well with the rest of the model; indeed, it seems the error may have arisen in changing this convention during modeling and failing to update all the affected rules. Why might such an error in the specification not be revealed in tests for these rules, especially if the test for each rule exercised transitions from the other rule also? Actually, had the error been reproduced in the implementation, it would have been found in these tests, since the visible system state would be obviously incorrect after both rules were traversed: the expected output for the tests was correct because, in constructing a test for a rule, one would naturally tend to refer to other rules only to check some point of uncertainty, not to confirm behaviour one knows to be correct. This was a rare case where, in inspecting each rule, it was not enough to know the broader system behaviour; one needed also to examine the detail of the other rule in proximity to it to realize that a change that might have happened in either rule was not happening at all.

Mistakes like this, and some less interesting ones, show that even careful application of a systematic, explicitly comprehensive approach cannot in itself eliminate human error arising from time pressure, boredom,

or distractions. That said, the errors in the CDR specification seem all to be "obvious" ones, none of which affected the software itself. Also, with the tool support now available, these mistakes would have been detected at the time. However, there can in general be no guarantee that an error in a model will cause it to be internally inconsistent or formally incomplete; and even an "obvious" error could transfer to an implementation through automatically-generated code. Ultimately, the only way to be absolutely sure of a perfect specification is to get a perfect specifier to write it.

In spite of having to check specification completeness and consistency manually, the approach taken to specify the CDR fulfilled everything required of it, both in terms of the end result and the relative efficiency of the process by which it was arrived at. The application code was eventually delivered on schedule, as was the final system. This was largely attributable to the very high quality of the specification, which had been identified at the outset as critical to the project's success.

4.2 Subsequent Projects

The commercial success of the CDR870 led to the development of other CDRs with more than one tray and substantially extended functionality. Some of these eclipsed the original both in terms of perceived complexity and according to more objective measures: to a good approximation, the first major variation was double the size of the original, both in terms of specification page count and lines of code; these figures doubled again with the next major addition of functionality. Though the author had no involvement in specifying these later systems, the tabular approach continued to be used. However, it was adapted (as outlined later) to make it more scalable.

While there was some turnover of personnel, experience with the approach on subsequent systems was broadly consistent with that on the first. Although it was described as requiring a "massive" effort initially, the detail and completeness of the resulting specifications were highly valued. In a document, one project leader credited the method with various benefits, including allowing improved estimation and preventing the implementation phase from becoming an iterative process of issue resolution which would be "much more time-consuming" and "very difficult to control." It was appreciated for similar reasons by the engineers working directly with it, one of whom, for example, described the specification as a very "safe" document from a contractual point of view.

The process by which the specifications actually got written was, as with the original system, one of periods of writing interspersed with meetings with whomever was responsible for deciding on the system behaviour. On one project, for example, these were daily two-hour meetings. In these sessions, any issues that had been elicited by modeling the behaviour using the method would be described in the form of mainly verbally-expressed scenarios, together with the best options the engineers had identified and the implications of each one. It was always easy for anyone familiar with the broad functionality of the system to move from a rule to a scenario in which some aspects of the system state might be replaced by preceding events, for example, "Suppose the tray is closed and the user presses the *play* button and then, before the beginning of the first track is found, the user presses..." A customer representative also referred to the original CDR870 specification as the "Use Cases"; a name which gained general currency for all the projects.

The importance of the constraint table was confirmed by accident in the first of these projects. The two engineers using the approach, to which the author had left no written method guidance, decided to work first only on the transition rules, planning to return later to the constraint table which they regarded as less important. This seemed to cause certain difficulties which led their project leader to suggest they give the constraint table priority. This resolved the problem and it was subsequently apparent to them that their original strategy "didn't really make sense" and that, without doing the constraint table, "you wouldn't have thought about [the system] properly."

The author's original motivation for the constraint table was partly to help to emphasize the semantics of the variables. Seeing, e.g., that it is possible for *Mode* to have the value *play* when *Tray* is *closing* makes it more obvious that *Mode* does not necessarily reflect the current system activity, but rather the selected one.

More importantly, the constraint table records in one place choices about the states which affect the behaviour. For example, one might decide that *Repeat* must be *no repeat* whenever the tray was open. This would mean that, in a transition in which the tray opens, the variable *Repeat* would have to be reset; it would also imply that there could be no response to an attempt to toggle this setting while the tray was open. Conversely, if this fact were not recorded in the constraint table then

it would make it hard to see whether the transition table was complete: if there are no rules for states in which the tray is open and *Repeat* is, e.g., *repeat track* then completeness depends on whether these states are in fact reachable from the initial power-on state. Besides being extremely difficult to determine manually from the many rules, the set of reachable states is necessarily dynamic and incorrect as long as the specification is incomplete, as well as being quite sensitive to any mistake in the rules.

This was the reason for the difficulty described earlier. The underlying principle is simply this: one must consider what the possible states of the system are before attempting to define the transitions between them. This is naturally enforced by other notations such as state transition diagrams – at least at the level at which the states and transitions are represented, i.e., the single-variable FSM. Of course, it is possible, indeed likely, that one may decide to change the constraints in light of issues which arise in the course of completing the transition table. This is not a problem; the important thing is to have a starting point and a working record of the static part of the model.

One engineer also remarked that he would like to have been able to include complex constraints in the constraint table. While it would be nice to have a systematic way to identify all constraints, rather just those involving pairs of variables, this does not seem to have been a significant issue. Although this may vary depending on the system, experience with the CDR projects would seem to suggest that complex constraints will typically comprise a relatively small proportion of the total. Further, by their nature they are weak in comparison to the two-variable constraints in the table, and so they are responsible for excluding an even smaller proportion of states from the set of those which are permitted.

Another problem was a degree of inertia observed by a project leader in getting started on writing the specification. This was a bit surprising, given that the method is designed to allow a naturally incremental approach by the progressive addition of variables and rules. In part, this may have been caused by a relative lack of stability in the desired functionality. However, a reported issue involved choosing variables. While variables like *Repeat* may be easy and obvious choices, sometimes a certain amount of judgment is needed to best explain the observable states without unnecessary technical detail. For example, to model the inevitable delay between pressing the play button and playback commencing, the CDR specifications abstracted the system's internal activity simply by giving a variable the value "searching".

Also, though it may not be obvious from a finished specification, there is occasionally a choice between using one variable or two to model some aspects of the system state. One choice leads to additional columns while the other, depending on the details of the behaviour, may make additional rules necessary (the opportunities for using "no change" being reduced). The problem is that reversing such a choice later can mean a lot of work in manually changing rules. This issue did not really arise on the first project partly because the knowledge gained in the initial attempt to model the system using Statecharts meant that there was, in effect, a good deal of foresight. For example, an argument could be made for combining *Tray* and *Mode* into a single variable – and this argument is stronger if one is not initially aware that pressing "play" when the tray is open causes the system to enter a "closing in order to play" state.

Choosing a good set of variables is of course an unavoidable part of any modeling approach of this kind and one that can benefit not only from skill and experience but also from hindsight; the difference with the transition table format is the high cost of changing one's choice of variables. It seems therefore that the best approach to this difficulty is to try to make the task of refactoring easier through software support.

Overall, the positive attitude to the notation among its users was remarkable considering the increased difficulty of manually checking the specifications for completeness – a task already a little daunting for the original system. The general feeling seemed to be that the value of the approach in ensuring that everything was thought of at the specification phase justified the heroic effort of the manual checks made necessary by the continued absence of tool support. However, as the size of the specifications grew to many hundreds of pages there was a recognition that this could not be sustained for larger systems, even with the adaptations described below.

4.3 Improving Scalability

Clearly, there are physical limits to a naive application of the approach: in particular, the width of the tables is restricted by the width of the page on which they must be printed. Where this limit is reached, the reintroduction of structure into the specification becomes a practical necessity. To illustrate how this was man-

aged in practice, we can consider a dual-tray CDR, that is, a CDR in which one tray – or loader[2] – has playback functionality only and the other may be used for playback or recording.

The engineers working on the first such project addressed the problem by specifying the behaviour of each loader in a separate transition table. Thus, each loader is modeled as a separate FSM. Functionality which spans loaders, for example, synchronized recording from one CD to another, is modeled with the help of commands (in effect, internal events) passed from one loader to the other, the responses to which are specified in the same way as those to external events.

This strategy may be seen as a compromise in which the familiar model of communicating FSMs is used as a higher-level structuring primitive for multi-variable FSMs specified using tables. Additionally, since the two loaders share a good deal of functionality, it was possible to put common responses in a third transition table, leaving only the behaviour unique to each loader in the two loader-specific tables – which may thus be seen as inheriting the behaviour of a notional basic CD player.

In fact, the structure used in the dual-tray CDR specifications is a little more subtle than merely communicating FSMs and inheritance. In particular, it was convenient to make some state information common to the tables for both loaders where the response of each depended on it – in effect, shared variables. It is interesting therefore to note the possibilities of using the natural join operator of relational algebra as a formal basis for structuring [13] (a similar idea with a different motivation is described in [14]).

On the first dual-tray CDR, the two loaders operated fairly independently and engineers were happy with the results of their approach. However, the functionality which spanned loaders grew on a later project so that there was more coupling between them. The structure, while obviously necessary, became less desirable. The behaviour was described as being more difficult to conceptualize and not as well "connected" as it would be in a single table. Compared to one big table, internal events were "subtle," more appropriate for the implementation. One engineer described the problem simply as having to look at more than one part of the specification to see the overall response.

These comments at least are further evidence that the underlying, unstructured approach of the method is a cognitively desirable one. This is also consistent with the difficulties complained of by users of Statecharts who wanted to know "the state of the whole system at once" [15] – exactly the view presented by the tabular notation in its basic form and that which the engineers wished as far as possible to preserve. It is possible that the problem could be alleviated by software, for example, by using multiple windows to show rules from different tables simultaneously.

While the introduction of structure made larger models tractable, the intimidating size of these later specifications became an issue for at least some customer representatives, who changed between projects, with one reportedly expressing the opinion that the tabular specification should be for the engineers only. Prototyping is one option that could help address such a difficulty – one which would also allow early usability evaluation, with the specification itself being simulated through a mock-up of the user interface; in the context of incremental specification, one can imagine a message "response not yet defined for this case" appearing for absent transition rules. Alternatively, one might deliver to stakeholders a document condensed from the tabular specification, including in it the more illuminating scenarios which were elicited. In either case, the detailed behaviour of the system would still be considered systematically by the specifiers who would raise and discuss any unusual scenarios with stakeholders during the specification process, even if the latter chose to give final approval on the basis of a tested or described sample of behaviour rather than a thorough review of the complete model.

5 Comparisons

The format of the transition table is similar to that described by Monk and Curry [8] as a tabular interface to Olsen's Propositional Production System (PPS) [16]. This is also FSM-based and the potential for using automated analysis to determine certain dynamic properties [17] has been explored and related to existing model checking technology [18]. A spreadsheet-based tool, called the Action Simulator, was created which allows a model to be executed – though this is demonstrational and somewhat limited by the possibilities afforded by spreadsheet macros. The basic approach has been exercised on a number of case studies but has remained relatively undeveloped.

[2] Strictly, the word 'tray' was used for the physical tray only, not the associated functionality.

Some notational differences are worth mentioning. Whereas the transition table format allows table cells to contain lists of variable values, in PPS and its tabular equivalent only a single value or an implicit "don't care" is possible. The increase in expressive power of being able to specify sets of states in the conjunctive normal form of the transition table is quite significant: for the CDR specifications, the number of rules otherwise required would have been many times greater, making Action Simulator-style tables practically unusable even for the original CDR870. On a more minor point, blanks are used to indicate "don't care" and "no change." Empty cells were also originally used on the CDR projects, though only on the target rows, i.e., to represent "no change" (the extracts above are modified ones); however, it seems better to be able to distinguish cells which are empty merely due to an interruption or an open question from "no change" cells.

The motivation for this research is quite different from that applying in the work described here. A primary concern is the usability of the system being specified and the possibilities for simulation and prototyping; model checking is used to evaluate such interaction-related properties as the number of steps needed to move from one state to another. In contrast to this work, explicit completeness and the systematic discovery of corner cases is not addressed or identified as an issue. Indeed, the semantics are inconsistent with such a task: the absence of a rule for a particular state-event combination implicitly means that the associated action is not possible in that state. (This is not the only semantic difference: Olsen also defines a priority ordering for rules in order to support an inheritance mechanism [16]; this author prefers semantics which allow each rule to be reviewed individually without having to take into account those following it – of which there may be many.) That nothing analogous to the constraint table was produced by this research can therefore be explained by the differing perspective and objectives as well as the lack of significant industrial application.

SCR [9] is another tabular approach which originated in an effort to improve the specification of an industrial system. It has the advantage of being relatively mature, with good tool support. An SCR specification is based on the Four-Variable Model [19] whereby requirements are documented by describing the variables which are monitored by the system, those which are controlled, and the relation required to exist between them. In SCR, this relation is expressed using tables, each of which describes how the value of a variable is derived in terms of other variables, which may also be internal variables. Variables may be FSMs and numeric variables are also permitted. The tables, which come in a variety of formats, may include not just lists of values but also expressions to indicate the calculation of the new value of a variable in different circumstances. SCR is output-driven in organization: given any controlled variable, one may work back through the specification to the monitored variables to see how its value is determined in all cases.

The method presented here differs in several respects from SCR. Though the Four-Variable Model is a general one, conforming to it for the CDR specifications would have meant defining the hardware and software interfaces in unnecessary and unwanted detail. For the tray, for example, one would have been concerned with how it was detected when the tray had closed, the activation of the motor used to open or close it, and so on. Even had this hardware not already existed, these interfaces would not have been a concern for the specifications described above; including them would also have made the document much more intimidating to a non-technical reader unconcerned with lasers, sensors, relays, rotational velocities and so on. For the CDRs, variables like *Tray* instead show the externally-visible response simply as a change in state at an appropriate level of abstraction. This is not to suggest that the Four-Variable Model is not suitable or effective in many instances. However, it seems likely to be less applicable where a rigorous treatment of the interface between the system and the world is inappropriate and the main goal is to describe user-perceptible behaviour in an accessible way.

SCR supports also the formal inclusion of numeric variables, which is especially useful for systems such as factory process control and aerospace, where variables like temperature and altitude may dominate in a description of the overall behaviour. In this respect, it is certainly more expressive than the notation introduced here. However, all effective notations have boundaries: SCR, for example, also is not suited to modeling more complex variables such as the sequence of tracks programmed to be played from a CD; and the specifications for the CDRs coped well with this boundary by the pragmatic and flexible device of adding descriptive notes to rules where necessary. One can also envision a more formal description of operations on non-finite state variables being attached to transition rules; for the CDRs, these aspects were relatively simple and the main objective was not to describe every detail of the

system with mathematical rigour but rather to ensure that nothing was overlooked.

Nevertheless, the notation is clearly more specialized than SCR and most others. While one may incorporate non-finite state aspects into the transition rules as described above, the advantages of the approach depend on the behaviour being substantially amenable to description as a finite state machine. If a system's behaviour is dominated by numeric or algorithmic elements then it would obviously be better to use another notation in the first place.

A more fundamental difference with respect to both SCR and other methods is specification structure and organization. Mainstream approaches based on the FSM model employ a decomposition into communicating FSMs, albeit with a varying or additional structuring mechanisms. In SCR, a network of dependencies defines a cycle-free order of evaluation; this is also true of SpecTRM-RL [20], another output-driven, mainly tabular approach designed for process control systems, and a successor to an earlier graphical/tabular approach called RSML [21]. In Real-Time Structured Analysis / Structured Design (RTSA/SD or SA/RT) methodologies [22, 23], FSMs communicate using internal events and are arranged into a hierarchy, the levels of which show the directed communication between either individual FSMs or lower level groups of FSMs. The Statecharts notation also allows directed communication between FSMs at higher levels but within blocks adds the concept of a hierarchy of states: separate FSMs can appear as "orthogonal components" within states; these may have an indefinite existence where they appear immediately within the single top-level state, or in effect be instantiated on entry to a lower level state. This application of hierarchy being somewhat incompatible with directed communication, broadcast events are used. In SDL (e.g., [24]), FSM-based processes may be grouped into blocks in a manner similar to RTSA/SD with directed communication including a semantics of event queueing; object-oriented structuring concepts are also available.

Despite their variety, all the usual approaches share a decomposition into single-variable FSMs, each showing the behaviour of a small part of the system at all times. In contrast, the approach dsecribed here, like the Action Simulator tables, decomposes the behaviour into rules, each showing the response of the whole system in some explicit set of cases. One advantage of this is simplicity: non-technical readers are able to see the system response directly, without having to understand structural elements such as internal events or to trace dependencies. It also means the desired overall behaviour is recorded incrementally rule by rule, removing any effort of constructing or modifying a model of interacting FSMs to produce the same effect. Most importantly, whereas the overall behaviour is normally seen only through the sample-based windows of testing, simulation and scenarios, the tabular format used to specify the CDRs allowed systematic review of the whole-system response in every possible system state.

Of course, a lack of scalability is a negative consequence of discarding the kind of structure used in other FSM-based notations. While the approach in its raw form was adequate to model the original CDR system, for larger systems one must inevitably re-introduce structuring primitives. Since one is then no longer seeing or considering the state of the whole system at once, this obviously entails compromising to some degree the advantages of the method in its basic form. As seen with later CDRs, the effectiveness of the approach in a particular system will then inevitably depend to some degree on how well it can be decomposed into relatively independent parts.

The constraint table is novel both as a process and a notational element. Typically, invariants are defined in a relatively ad-hoc way when using a model checker to check a completed FSM model; the model is progressively corrected and the invariants strengthened as necessary until verified. The constraint table, in contrast, is designed to systematically circumscribe the states of the system as part the modeling process. While this is necessary for a notation requiring one to consider all possibilities, it also makes it possible to check that each rule does not allow a state not permitted by the constraints to be reached from one that is – and this check can be carried out independently on each rule as it is defined, before the model is complete. As a kind of logical spreadsheet, the constraint table format is easier to read and works better as a reference than an equivalent list of formulas; nothing is lost by having to keep as formulas those constraints that cannot be represented in the table, especially where these are relatively few in number.

6 Conclusions

Apart from the specific method, this work also contributes to a growing body of reports on the successful application of formal methods in practice. In this respect, it is worth noting that the principal motivation for

using a formal method of specification was to develop a product as quickly as possible. Outside of critical domains like aviation, the industrial adoption of formal methods is likely to be advanced by an increased emphasis on such obvious commercial benefits as reduced time-to-market.

The most important advantage of the method itself is the systematic elicitation of unusual scenarios, which, for the CDR projects, proved invaluable. With other notations, mathematical completeness is in general no indication that one has thought of everything; however, with the tabular notation, correctness tends to follow naturally from model completeness. In effect, the method creates a formal and complete model of system behaviour but one that is comparable to a description based on sequential scenarios, in that the model shows the response of the whole system at once, and does this in an equally immediate, straightforward way which reliably captures the intentions of the specifier.

It is also very much an incremental method, allowing new information to be incorporated in the appropriate place as it becomes available, progress and remaining work to be easily assessed, and correctness to be built in as part of the specification process. In this respect, it works because of the organization of the specification and the comparative lack of structure in the model: while the benefits of structure are well-understood, the primary concern at specification is how information may be recorded in an efficient way that avoids error.

In its basic form, the method seems to satisfy the objectives of simplicity and formality emphasised by, for example, Wieringa, who recommends defining "the simplest technique possible that still is of use", noting that a "simple core notation could be incrementally extended to suit the needs of the specifier" [25]. For the later CDR projects described above, this is essentially what happened when the engineers themselves incorporated structuring concepts from other approaches, without direction from the author. Nevertheless, making the method more suitable for larger systems requires further work in adapting and formalizing the kind of structuring techniques that seem best suited to it, while, in applying the method, the challenge is to introduce structure in such a way that the tabular components chosen are those for which the format will provide most benefit.

Despite its limitations, the method is one which has already proven practical and effective in producing high-quality, understandable specifications and shortening the development lifecycle. Its principal domain of applicability would appear to be embedded interactive systems with substantial finite-state aspects, that is, systems likely to manifest the same kind of complexity as the CDRs described above. In critical domains, the value of a specification method designed to ensure that no exceptional cases are overlooked would obviously assume greater significance. A user guide to the notation has been written and a supporting software tool, called Statestep, is also available [26].

Acknowledgments

CDR project data is included by kind permission of Joop Kerssen, Philips. The anonymous reviewers suggested several changes which improved this article. The author is indebted to Dan Berry for his advice and enthusiasm. Special thanks for their help and co-operation are also due to: Marc Cools and Martine Looymans at Philips; Stefano Fontolan, John Mulcahy, and the engineers at Silicon & Software Systems; Norah Power.

References

[1] D. Jackson and J. Wing, "Lightweight formal methods," *IEEE Computer*, Apr. 1996.

[2] C. Potts, "Software engineering research revisited," *IEEE Softw.*, pp. 19–28, Sept. 1993.

[3] D. Parnas, "Successful software engineering research," *ACM SIGSOFT Software Engineering Notes*, vol. 23, no. 3, pp. 64–68, May 1998.

[4] M. Lubars, C. Potts, and C. Richter, "A review of the state of the practice in requirements modeling," in *Proceedings of the Interational Requirements Engineering Symposium*. Los Alamitos, CA: IEEE Computer Society Press, 1992, pp. 2–14.

[5] B. L. Kovitz, *Practical Software Requirements*. Greenwich, Connecticut: Manning, 1998.

[6] N. D. Murphy, *Front Panel: Designing Software for Embedded User Interfaces*. Lawrence, KS: R&D Books, 1998.

[7] M. K. Zimmerman, K. Lundqvist, and N. Leveson, "Investigating the readability of state-based formal requirements specification languages," in

Proc. ICSE'02 24th International Conference on Software Engineering, 19–25 May 2002.

[8] A. F. Monk and M. B. Curry, "Discount dialogue modelling with Action Simulator," in *People and Computers IX: Proceedings of HCI'94*. Cambridge University Press, 1994.

[9] C. L. Heitmeyer, R. D. Jeffords, and B. G. Labaw, "Automated consistency checking of requirements specifications," *ACM Transactions on Software Engineering and Methodology*, vol. 5, no. 3, pp. 231–261, July 1996.

[10] D. Harel, "Statecharts: a visual formalism for complex systems," *Science of Computer Programming*, vol. 8, pp. 231–274, July 1987.

[11] D. Parnas and P. Clements, "A rational design process: How and why to fake it," *IEEE Trans. Software Eng.*, vol. SE-12, no. 2, pp. 251–257, Feb. 1986.

[12] D. Berry, K. Daudjee, J. Dong, I. Fainchtein, M. Nelson, and T. Nelson, "User's manual as a requirements specification: Case studies," *Requirements Engineering Journal*, vol. 9, pp. 67–82, 2004.

[13] A. ten Berg, C. Huijs, and T. Krol, "Relational algebra as formalism for hardware design," *Microprocessing and Microprogramming*, vol. 38, pp. 403–410, 1993.

[14] T. T. Lee and M.-Y. Lai, "A relational algebraic approach to protocol verification," *IEEE Trans. Software Eng.*, vol. 14, no. 2, pp. 184–193, Feb. 1988.

[15] J. C. Knight, C. L. DeJong, M. S. Gibble, and L. G. Nakano, "Why are formal methods not used more widely?" in *Proc. LFM97: Fourth NASA Langley Formal Methods Workshop*, Hampton, VA, 10–12 Sept. 1997.

[16] D. R. Olsen, "Propositional production systems for dialog description," in *Human Factors in Computing Systems: Proceedings CHI'90*. ACM Press, 1990, pp. 57–63.

[17] D. R. Olsen, A. F. Monk, and M. B. Curry, "Algorithms for automatic dialogue analysis using propositional production systems," *Human-Computer Interaction*, vol. 10, pp. 39–78, 1995.

[18] G. D. Abowd, H.-M. Wang, and A. F. Monk, "A formal technique for automated dialogue development," in *Proc. Designing Interactive Systems: Processes, Practices, Methods and Techniques*, Ann Arbor, MI, 23–25 Aug. 1995.

[19] D. L. Parnas and J. Madey, "Functional documents for computer systems," *Science of Computer Programming*, vol. 25, pp. 41–61, 1995.

[20] N. G. Leveson, M. P. Heimdahl, and J. D. Reese, "Designing specification languages for process-control systems: Lessons learned and steps to the future," in *Proc. ESEC/FSE '99, 7th European Software Engineering Conference held jointly with the 7th ACM SIGSOFT Symposium on the Foundations of Software Engineering*, 6–10 Sept. 1999.

[21] N. G. Leveson, M. P. Heimdahl, H. Hildrith, and J. D. Reese, "Requirements specification for process-control systems," *IEEE Trans. Software Eng.*, vol. 20, no. 9, pp. 684–707, Sept. 1994.

[22] P. T. Ward and S. J. Mellor, *Structured Development for Real-Time Systems*. Englewood Cliffs, New Jersey: Prentice Hall, 1985.

[23] D. J. Hatley and I. A. Pirbhai, *Strategies for Real Time System Specification*. New York: Dorset House, 1988.

[24] J. Ellsberger, D. Hogrefe, and A. Sarma, *SDL: Formal Object-oriented Language for Communicating Systems*. Prentice Hall, 1997.

[25] R. Wieringa, "A survey of structured and object-oriented software specification methods and techniques," *ACM Computing Surveys*, vol. 30, no. 4, Dec. 1998.

[26] M. Breen, "Statestep: A tool for systematic, incremental specification," in *Proceedings of the 26th International Conference on Software Engineering (ICSE'04)*. IEEE Computer Society, 23–28 May 2004, pp. 711–712.