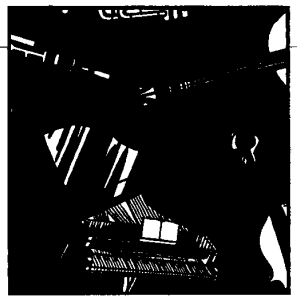

A Formal Specification of an Oscilloscope

Norman Delisle and **David Garlan**, Tektronix

Unlike most work on the application of formal methods, this research uses formal methods to gain insight into system architecture. The context for this case study is electronic instrument design.

Ten years ago, electronic instruments such as oscilloscopes and spectrum analyzers consisted almost entirely of analog hardware controlled by hardwired knobs and switches. However, in the last decade, software has come to dominate the implementation of electronic instruments.

A modern oscilloscope's software component may consist of more than a megabyte of code providing features such as complex measurements, waveform storage, and control via a local-area network. Its internal architecture typically includes multiple microprocessors running real-time operating systems, special-purpose coprocessors for digital signal processing, and user interfaces that support windows, pop-up menus, and soft knobs and switches.

The incorporation of software technology into electronic instruments happened so rapidly that software engineers in this area have not had a rich history of theory and practice to draw on. Of course, electronic instrument developers aren't alone: This problem faces developers of many contemporary embedded systems. In contrast, a software engineer developing a compiler can base his designs on a well-accepted top-level architecture and considerable supporting detail in areas such as algorithms and data structures for parsing and semantic analysis.

Over the last three years, we have conducted in-depth studies to determine how formal methods can help us deal with these problems of increasing software complexity. This case study describes one of the outcomes of our research. It illustrates how oscilloscope developers can use formal methods to express a reusable architecture analogous to the models available to compiler developers.

While there has been considerable interest recently in applying formal methods to industrial software development,[1-3] our focus differs from most other work. Most applications of formal methods tend

0740-7459/90/0900/0029/$01.00 © 1990 IEEE

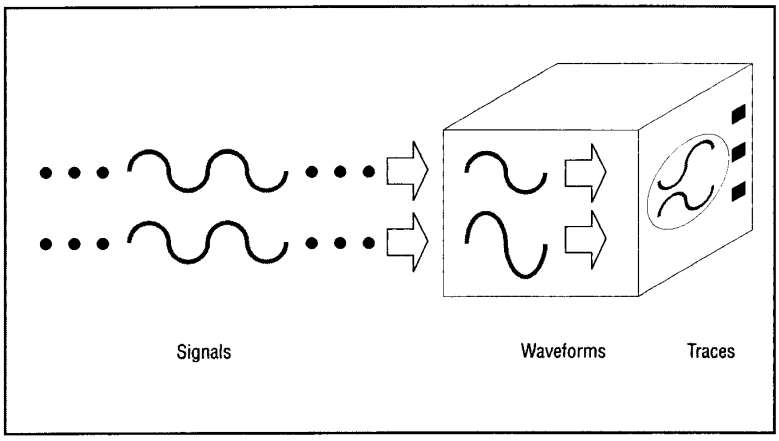

Figure 1. A simple view of an oscilloscope.

to emphasize correctness and formal refinement.[4] In contrast, we focus on using formal specifications *to gain insight* into a system's required behavior. In this context, specifications play an important role in clarifying the design and in leading to a good problem decomposition.

This case study presents the development of an abstract oscilloscope specification. Our specification uses Z notation. If you are not familiar with Z, you should be able to get enough information from the context to get an intuitive idea of what our specification does. The box on p. 34 is a brief introduction to Z and a glossary of symbols we used in this specification.

Problem and context

Put simply, an oscilloscope is a hardware/software system that displays pictorial representations of time-varying electrical signals. As Figure 1 illustrates, electrical signals come into the oscilloscope through one or more channels and are converted into waveforms. After some internal processing, the waveforms are displayed as pictures called *traces*.

Several factors complicate this simplified view. First, an oscilloscope user is typically interested in only certain aspects of a signal, such as its periodic nature, its form over some small time interval, or its deviation from some other signal. So the user must have some way to *acquire* the signal portions he is interested in. Typically, this is done with a *trigger*, which determines when waveform acquisition should occur. The trigger could be obtained from one of the signal channels or from an external signal source.

A second complication is that oscilloscopes are limited in their ability to acquire and display a signal faithfully. To get the best performance out of an oscilloscope, the user may have to adjust several parameters so, for example, the signal falls within the appropriate dynamic range of acquisition hardware and so the waveform fits on the small display.

A third complication is that an oscilloscope usually can do many things besides displaying waveforms as traces: It can measure the waveform (for example, rise time), store and retrieve waveforms, perform arithmetic operations on waveforms (for example, adding two waveforms), display simultaneous views of one or more waveforms, and interact with external devices (like workstations). Indeed, there may be more than 300 user-level commands for configuring and operating an oscilloscope.

The complexity of today's oscilloscopes demands clear user-level models for understanding how an oscilloscope functions. Toward this end, we have been working to develop formal specifications of such models. Beyond the obvious need to satisfy the requirements of cleanliness, clarity, and correctness, the primary challenge in specifying an oscilloscope is to provide a reasonable treatment of its dynamic behavior.

At first it may seem that a simple functional approach might suffice, since, after all, an oscilloscope basically transforms signals into traces. However, there are three very different forms of dynamic behavior that the specification must deal with. The first is the dynamic input signal. The second is the dynamic acquisition of waveforms, which generates a sequence of traces over time. And the third is the dynamic oscilloscope configuration, which determines how, when, and where the signal is acquired and displayed.

An abstract oscilloscope

Our goal was to provide a users' abstract model of an oscilloscope to clarify its user-accessible functions. (We've described elsewhere how a similar specification yields a reusable framework.[5]) Our model is abstract in two ways: It suppresses the functions' implementation details and is independent of any user interface. For example, our model neither indicates which functions are implemented in hardware and which in software, nor tries to explain how those functions are controlled by specific knobs, menus, or buttons.

Our plan of attack was to adopt a simple functional view: We described an oscilloscope as a mathematical function applied to a signal to produce a series of traces. We handled the intrinsic complexity of the mathematical function by treating it as a composition of simpler functions, each of which describes part of the oscilloscope process.

To account for a user's ability to configure an oscilloscope, we treated the oscilloscope building blocks as higher order functions. Thus, given a set of user-provided parameters, an oscilloscope component produces a new function that performs some transformation on its input data.

Higher order functions nicely model both the user's intuition about how to configure an oscilloscope and the actual operating behavior of typical oscilloscope implementations. Typically, the user sets up values for input parameters and then the oscilloscope repeatedly applies the resulting functions in real time to produce a series of traces. Traces for periodic signals give the illusion of being live because new trace values rapidly overwrite nearly identical old values.

The specification. To illustrate our approach, we specify a very simple oscilloscope: It has only two channels and provides no advanced functions like measurement or storage.

Data types. In Z, data types are sets of values. We model our notions of time — both absolute and relative — as natural numbers ($\mathbb{N}$) and voltage and screen coordinates in terms of integers ($\mathbb{Z}$). (Both natu-

ral numbers and integers are predefined in Z.) Thus, we have five basic types:

$AbsTime == \mathbf{N}$
$RelTime == \mathbf{N}$
$Volts == \mathbf{Z}$
$Horiz == \mathbf{Z}$
$Vert == \mathbf{Z}$

We choose to base the time domain on natural numbers instead of real numbers because an oscilloscope has a finite bandwidth, and so it has finite time resolution. You can think of AbsTime as the number of time ticks since some absolute reference time (say, nanoseconds since the invention of the oscilloscope). Similarly, you can think of RelTime as the number of ticks since a relative reference time (say, nanoseconds since a particular trigger event). We used a discrete definition of volts for a similar reason — an oscilloscope has a finite voltage resolution due to the thermal noise of its electrical components.

We define the data types Signal, Waveform, and Trace as mathematical functions:

$Signal == AbsTime \rightarrow Volts$
$Waveform == AbsTime \leftrightarrow Volts$
$Trace == Horiz \leftrightarrow Vert$

A signal is modeled as a function from time to voltage. This definition abstracts away the passage of time: You can observe the signal's entire past and future at any point in real time. A waveform is obtained by extracting a bounded time slice of a signal. Thus, we model a waveform as a partial function from time to voltage. (That is, the function is defined for only some values of its domain.)

A trace is a graphical view of a waveform. To allow for clipping when displayed on a finite-sized display, we define the trace as a partial function in the Horiz-Vert coordinate space. Figure 2 illustrates the relationship between signals, waveforms, and traces.

Channels. A channel represents the path along which a signal is converted to a trace; it is modeled as a function from signals to traces. We define a channel's behavior as a composition of higher order functional components. Figure 3 shows a road map for this part of the specification, which is decomposed into components

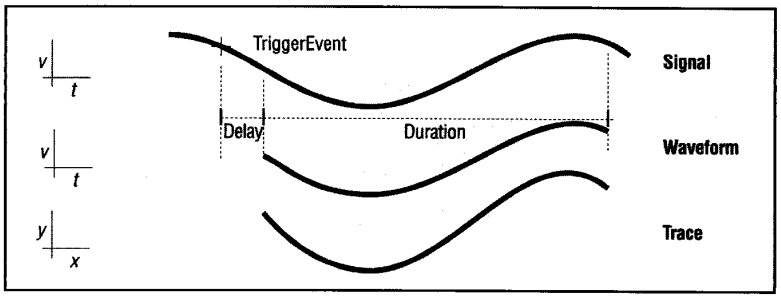

Figure 2. Signals, waveforms, and traces.

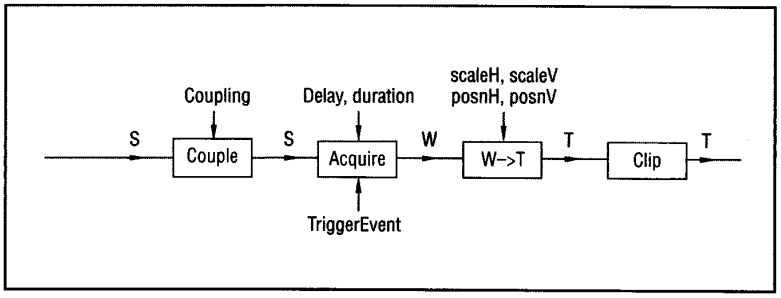

Figure 3. Oscilloscope channel.

that handle *coupling, acquisition, scaling,* and *clipping.*

A typical oscilloscope processes the incoming signal in various ways before waveforms are extracted from it. As an example of this kind of function, we define a function to subtract a DC offset from a signal. The specification offers three choices: DC, AC, and GND. If the user chooses DC coupling, the signal is passed along unaltered; if AC, the dc function subtracts the DC voltage component; and if GND, the operation is a constant function that yields zero volts for all time values:

$Coupling ::= DC \mid AC \mid GND$

$Couple : Coupling \rightarrow Signal \rightarrow Signal$
$dc : Signal \times AbsTime \rightarrow Volts$

$Couple\ DC\ s = s$
$Couple\ AC\ s = (\lambda t : AbsTime \bullet s(t) - dc(s, t))$
$Couple\ GND\ s = (\lambda t : AbsTime \bullet 0)$

These Z fragments define the type Coupling as a disjoint union of the constants DC, AC, and GND. Couple and dc are defined as global functions, constrained by the predicates listed below the horizontal line, which effectively define Couple for each coupling option.

We have omitted the definition of the dc function, which computes a signal's average DC component at any point in time. The Couple function is typically implemented as a simple high-pass filter, and its design is well-understood. If you intended to use this abstract model to compute the signal transformation precisely, you would need a more detailed mathematical description.

In writing a formal specification for a system as complex as an oscilloscope, it is important to focus on the questions that you seek to answer and omit irrelevant details. In the Z specification for the Couple operation, it may seem as if we have left out the hard part — the definition of dc — and gone to great pains to be precise about the easy parts.

The justification for this approach is that we are more interested in defining an abstract framework than in specifying the low-level details of an oscilloscope. This framework specifies oscilloscope operations as higher order functions that are applied to user-specified parameter values and then composed. It is important to be able to show that all functions of an oscilloscope fit into this model, but the details of many of those functions — such as Couple — may not be interesting. Further, we gain more significant insight when we specify more complex oscilloscope operations, such as Acquire.

The Acquire operation lets the user select the aspect of the signal he is interested in. In effect, it converts a signal to a wave-

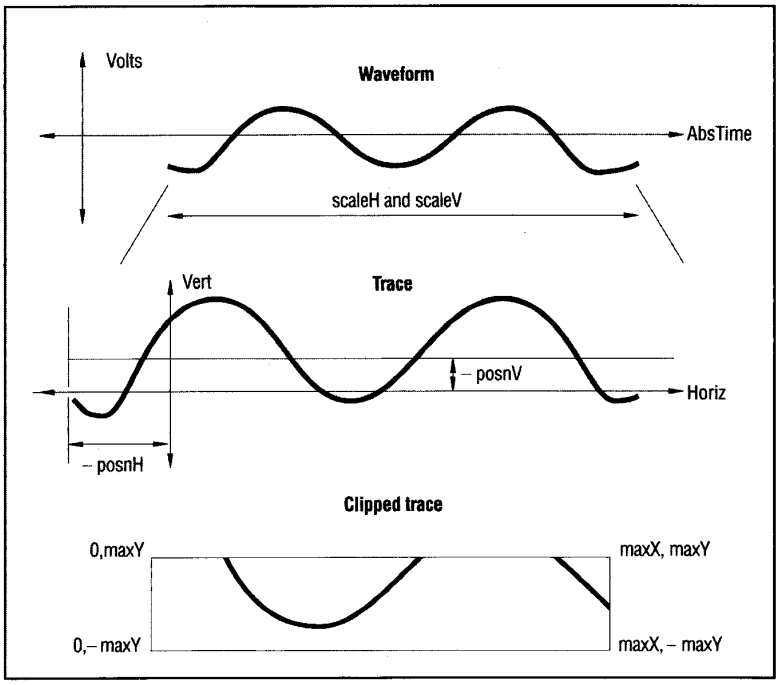

Figure 4. Converting a waveform to a trace.

form by extracting a time slice. The waveform is identical to the original signal except that it is defined only for the time values within a bounded interval. A TriggerEvent is a reference point for that interval; two RelTime values are supplied to specify a duration and delay relative to the TriggerEvent, as Figure 2 illustrates.

$$TriggerEvent == AbsTime$$

$$\begin{array}{|l} Acquire: RelTime \times RelTime \\ \quad \rightarrow TriggerEvent \\ \quad \rightarrow Signal \\ \quad \rightarrow Waveform \\ \hline Acquire\ (delay, dur)\ trig\ s = \\ \quad \{t: AbsTime \mid trig + delay \leq t \leq \\ \quad\quad trig + delay + dur\} \lhd s \end{array}$$

A waveform is converted into a trace in two steps, as Figure 4 illustrates. The first step comprises the scaling and positioning component of the channel specification; the second comprises the clipping component.

The scaling and positioning component is embodied in the WaveformToTrace function, which performs a domain conversion by converting each time/voltage pair into a pair of horizontal and vertical values. The WaveformToTrace function also determines the horizontal and vertical axes.

The nominal conversion (a zero-valued posnH and posnV) aligns both the minimum time point in the waveform with the horizontal origin and the ground voltage level with the vertical midpoint.

$$\begin{array}{|l} WaveformToTrace: \\ \quad RelTime \times Volts \times Horiz \times Vert \\ \quad \rightarrow Waveform \rightarrow Trace \\ \hline WaveformToTrace\ (scaleH, scaleV, \\ \quad posnH, posnV)\ w = \\ \quad \{t, t1: AbsTime; v: Volts \mid \\ \quad\quad t \mapsto v \in w \wedge t1 = t - min\ (\mathrm{dom}\ w) \bullet \\ \quad\quad\quad TimeToHoriz(t1, scaleH, posnH) \\ \quad\quad\quad\quad \mapsto VoltsToVert\ (v, scaleV, posnV)\} \end{array}$$

The WaveformToTrace function uses the TimeToHoriz and VoltsToVert functions to convert individual time and voltage values. The parameters scaleH and scaleV establish the calibration of the graphical coordinates with respect to time and voltage. The horizontal and vertical offset positions, posnH and posnV, pan through the coordinate space.

$$\begin{array}{|l} maxH: Horiz \\ maxV: Vert \\ TimeToHoriz: \\ \quad AbsTime \times RelTime \times Horiz \rightarrow Horiz \\ VoltsToVert: Volts \times Volts \times Vert \rightarrow Vert \\ \hline TimeToHoriz\ (t, scaleH, posnH) = \\ \quad (t * maxH)\ \mathrm{div}\ scaleH + posnH \\ \\ VoltsToVert\ (v, scaleV, posnV) = \\ \quad (v * maxV)\ \mathrm{div}\ scaleV + posnV \end{array}$$

The second step in converting a waveform to a trace clips the trace to fit into a display screen of width maxH and height maxV. The Clip function excludes points in the trace that lie outside the screen's display space. (Clipping is expressed succinctly in Z using both domain and range restriction operators, as the box on p. 34 describes.)

$$\begin{array}{|l} Clip: Trace \rightarrow Trace \\ \hline Clip\ tr = (0\ ..\ maxH) \lhd tr \rhd (0\ ..\ maxV) \end{array}$$

Now we combine the components of the channel subsystem: coupling, acquisition, scaling, and clipping. The channel configuration state is explicitly defined as a collection of user-supplied parameters for a channel. We encapsulate those parameters in a Z schema and then, to get the channel subsystem, we apply the previously defined higher order functions to those parameters and compose the resulting functions.

$$\begin{array}{|l} ChannelParameters \\ \hline c: Coupling \\ delay, dur: RelTime \\ scaleH: RelTime \\ scaleV: Volts \\ posnV: Vert \\ posnH: Horiz \end{array}$$

$$\begin{array}{|l} ChannelConfiguration: \\ \quad ChannelParameters \rightarrow TriggerEvent \\ \quad \rightarrow Signal \rightarrow Trace \\ \hline ChannelConfiguration\ p = (\lambda\ trig: \\ \quad TriggerEvent \bullet \\ \quad Clip \circ WaveformToTrace(p.scaleH, \\ \quad\quad p.scaleV, p.posnH, p.posnV) \\ \quad\quad \circ Acquire\ (p.delay, p.dur)\ trig \\ \quad\quad \circ Couple\ p.c) \end{array}$$

The resulting ChannelConfiguration function neatly bundles each component function of the channel subsystem to get yet another higher order function. In this case, of course, the higher order function takes the entire channel configuration state to produce a trace from a signal.

Triggers. A waveform is acquired in response to a TriggerEvent. Figure 5 illustrates the trigger part of the specification.

A TriggerEvent is an AbsTime value that corresponds to the occurrence of an interesting feature on a signal. The user controls the parameters to define "interesting." However, the first thing a user must do is pick the channel from which trigger events are selected. SelectChannel

accomplishes this:

$Channel ::= CH1 \mid CH2$

$SelectChannel : Channel \rightarrow Signal \times Signal \rightarrow Signal$

$SelectChannel\ CH1\ (s_1, s_2) = s_1$
$SelectChannel\ CH2\ (s_1, s_2) = s_2$

After selecting a signal source, the user selects the coupling with the Couple function. We can use the coupling function defined earlier. However, typical oscilloscopes do not let the trigger coupling value be GND because a constant zero-valued function could not be used for a trigger. We show later how this restriction is enforced.

When the user has selected a signal source and conditioned that signal, he can look for TriggerEvents. In this simple case, the user defines an interesting signal event as one for which the signal crosses a specific voltage level along a positive or negative slope. To specify this, we first introduce the types Slope and Level and then define the function DetectTrig, which produces a set of time values representing the extracted trigger events.

$Slope ::= POS \mid NEG$
$Level == Volts$

$DetectTrig : Level \times Slope \rightarrow Signal \rightarrow \mathbf{P}\ TriggerEvent$

$\forall t : DetectTrig\ (l, sl)\ s \bullet$
$((sl = POS) \Rightarrow s(t-1) \leq l \leq s(t)) \wedge$
$((sl = NEG) \Rightarrow s(t) \leq l \leq s(t-1))$

This definition does not say that the result of DetectTrig contains every point on the signal with the given slope and level, only that every point selected has those properties. Such nondeterministic functions are particularly useful for leaving details of the model open for future refinements.

In this case, it turns out that some oscilloscopes ignore some of the points on the trigger signal, so the predicate on DetectTrig is just what we want. (This specification does, however, leave open the possibility that the oscilloscope might produce no traces.)

We then put the pieces together for the Trigger subsystem by applying the operations to the user-supplied parameters and composing the component functions.

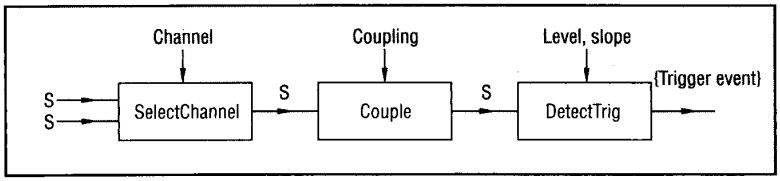

Figure 5. Trigger selection.

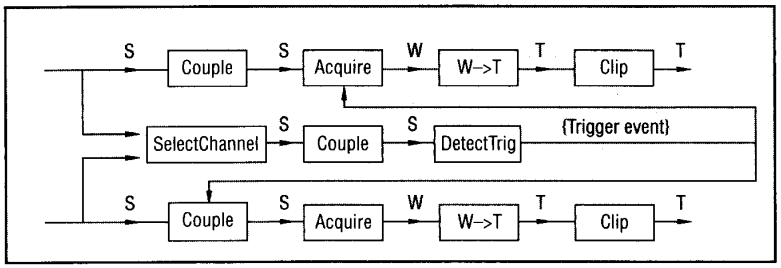

Figure 6. The whole oscilloscope.

TriggerParameters

$l : Level$
$sl : Slope$
$ts : Channel$
$tc : Coupling$

$tc \in \{AC, DC\}$

$TriggerConfiguration :$
$TriggerParameters \rightarrow Signal \times Signal$
$\rightarrow \mathbf{P}\ TriggerEvent$

$TriggerConfiguration\ p =$
$DetectTrig\ (p.l, p.sl) \circ Couple\ p.tc$
$\circ SelectChannel\ p.ts$

This subsystem specification reuses the Couple function but constrains the values of its input parameter — in the predicate of the TriggerParameters schema — to AC and DC only.

The overall oscilloscope. With these specifications in place, we can put all the pieces together. Figure 6 shows the architecture: Two channels, each with its own set of parameters, produce traces as defined by trigger events selected from one of these channels.

The following specification captures this architecture formally. It relates each trace in the displayed sequence to the TriggerEvent used to acquire it. It does not require that every trigger produce a trace, only that every trace must have been produced by a trigger. The overall oscilloscope is defined as

Oscilloscope

$s1, s2 : Signal$
$cp1, cp2 : ChannelParameters$
$tp : TriggerParameters$
$ts1, ts2 : \text{seq}\ Trace$

$\forall t : \text{ran}\ ts1 \bullet$
$\exists trig : TriggerConfiguration\ tp\ (s1, s2) \bullet$
$t = ChannelConfiguration\ cp1\ trig\ s1$

$\forall t : \text{ran}\ ts2 \bullet$
$\exists trig : TriggerConfiguration\ tp\ (s1, s2) \bullet$
$t = ChannelConfiguration\ cp2\ trig\ s2$

Scaling up

In scaling up this specification to deal with more complicated, realistic oscilloscopes, you must address three issues.

First, you must describe the instrument's electrical properties and limitations more accurately. An instrument is an electronic circuit and so has many physical limitations. A formal oscilloscope model must describe the details of electrical behavior. For example, oscilloscope designers are concerned with phenomena like time skew and jitter, bounded frequency response, and both voltage and time calibration inaccuracies. Our specification defines an idealized oscilloscope that you can further constrain to specify a real oscilloscope that takes these limitations into account. Hayes has done just that by extending our work in a similar

Using Z

In addition to the basic Z types defined on p. 25, we used two additional kinds of type definitions in our oscilloscope specification. A simple type definition is written

$X == Y$

which defines X as an abbreviation for Y.

An enumerated type definition is written in the form:

$X ::= C1 \mid C2 \mid C3$

This defines X to be a set of three atomic values: $C1$, $C2$, and $C3$.

The following define the symbols used in this article. In the symbols' definitions, S, X, and Y are sets; for example, $x:X$, $y:Y$. R is a binary relation; for example, $R: X \leftrightarrow Y$. Last, f and g are functions; for example, $f,g: X \rightarrow X$.

$\mathbb{Z}$	The set of integers.
$\mathbb{N}$	The set of natural numbers.
$X \rightarrow Y$	The set of total functions from X to Y.
$X \nrightarrow Y$	The set of partial functions from X to Y.
$\lambda x: X \bullet f(x)$	The function that, given an argument x of type X, produces the result $f(x)$.
$X \times Y$	Cartesian product.
$x \mapsto y$	x maps to y.
$S \lhd R$	Domain restriction: $\{x \mapsto y \mid x \in S \wedge (x \mapsto y) \in R\}$.
$R \rhd S$	Range restriction: $\{x \mapsto y \mid y \in S \wedge (x \mapsto y) \in R\}$.
$f \circ g$	Functional composition.
min S	The minimum value of the set S.
dom R	The domain of the relation R.
ran R	The range of the relation R.

Is Z an appropriate notation for the industrial use of formal methods? Overall, we have found Z well suited to our needs. In particular, its ability to define both deterministic and nondeterministic functions, coupled with the ability to encapsulate state, is sufficient for a functional style of formal description, such as the one illustrated in this article.

This approach is loosely related to functional decomposition as a design methodology, since it views the overall oscilloscope-design problem as one of decomposing a complex function into many simpler functions. On the other hand, it differs considerably insofar as those functions are basic, interchangeable, reusable components that can be composed into more complex instruments. In other specifications, we have relied more heavily on the use of schemas and Z's schema calculus. We have written elsewhere about alternative ways to use Z to model an oscilloscope and Z's drawbacks.[1]

Reference

1. D. Garlan and N. Delisle, "Formal Specifications as Reusable Frameworks," *VDM 90: VDM and Z!*, Springer-Verlag, New York, 1990, pp. 150-163.

formal model of an oscilloscope.[6]

Second, your specification must be able to describe a rich repertoire of dynamically configurable oscilloscope operations. We crafted our model to aid reasoning about many configurations of oscilloscope operations. Our simple oscilloscope has a small repertoire of functions arranged in a fixed configuration — its operations are limited to modifying parameter and input-signal values.

However, more sophisticated oscilloscopes let you change the channel configuration and the trigger system. For example, the user should be able to add a measurement as an operation on a waveform. There are two ways to handle support for dynamic configurations: You can define an all-encompassing static configuration and disable the operations not in use, or you can provide explicit reconfiguration operations. We took the former approach in our simple oscilloscope. Because the DC setting of Couple leaves a signal unchanged, we might have dynamically reconfigured the channel without the Couple component. However, defining all possible configurations becomes unwieldy as the operations repertoire grows.

Fortunately, the second solution — providing reconfiguration operations — is straightforward. You can describe a channel's primary flow statically; new operations tend to branch off at well-defined points. In the most general case, an oscilloscope's channnels are a directed acyclic graph whose vertices represent operations and edges represent dataflow. The directed acyclic graph representation is powerful enough to describe interchannel operations, such as the addition of two waveforms.

Third, a more realistic specification needs more and richer types. Types are needed for measurements. Furthermore, you would have to enhance the definition of waveforms. Defining a waveform as a function (as we did) turns out to be insufficient in more complex oscilloscopes because there are special waveform types that require more than one voltage value for each time-domain element.

Gaining insight

Success in specification writing comes from knowing what to include and what to ignore. By taking a simplified view of the passing of time, this specification becomes simpler and more elegant. However, it also limits the questions the formal specification can answer.

In this case, the specification does not clearly describe what happens when the user modifies a parameter. For example, the specification gives the illusion that you can reacquire a waveform whose acquisition window has already passed.

Abstracting away from time turns out to be a useful approach because there is much to gain from a steady-state description of oscilloscope behavior. An alternative, more detailed formal model could be developed to explore some dynamic issues.

User's view. This specification defines a user's model of an oscilloscope's internal operations. We chose to model the user's perspective because an electronic instrument is an interactive system that needs a simple, clean conceptual model. We also wanted to transcend the biases that arise from intimate knowledge of how instruments are implemented. All too early in the design process, engineers tend to think about concrete issues such as hardware/software trade-offs and data representations.

We maintained a high abstraction level by constantly asking, "Does the user need to know about that?" Because we chose the user's perspective, the specification can be used both as a basis for design and for user-level descriptions such as user manuals.

Formal models. Our work focuses on using formal notations for gaining insight into systems. We use mathematical tools

and models to clarify how a system works so we can produce better system designs. Thus, for us formal models are more interactive tools for posing questions and suggesting solutions than they are a way to document a polished product. In this respect, formal specifications function as conceptual prototypes that you ask formal questions of and reason about, much as you would experiment with a prototype.

To illustrate how a formal model can lead to increased insight, consider arithmetic operations in an oscilloscope. Suppose you want to subtract one signal from another to examine their differences. There are three types in our oscilloscope model on which you can define subtraction: It could operate on two signals, two waveforms, or two traces.

$$\begin{array}{l} SubtractS: Signal \times Signal \rightarrow Signal \\ SubtractW: Waveform \times Waveform \\ \quad \rightarrow Waveform \\ SubtractT: Trace \times Trace \rightarrow Trace \end{array}$$

$$\begin{array}{l} SubtractS(s1, s2) = \lambda\, t : \mathrm{dom}(s1) \\ \quad \mid \mathrm{dom}(s1) = \mathrm{dom}(s2) \bullet s1(t) - s2(t) \\ SubtractW(w1, w2) = \lambda\, t : \mathrm{dom}(w1) \\ \quad \mid \mathrm{dom}(w1) = \mathrm{dom}(w2) \bullet w1(t) - w2(t) \\ SubtractT(t1, t2) = \lambda\, x : \mathrm{dom}(t1) \\ \quad \mid \mathrm{dom}(t1) = \mathrm{dom}(t2) \bullet t1(x) - t2(x) \end{array}$$

Subtracting signals and waveforms yields similar results: At each time in either domain the result is the voltage difference between the two operands. Trace subtraction is performed on graphics objects, so the result depends on graphical scaling or positioning operations performed on either of the operands.

All three subtraction types could be useful; indeed, different oscilloscopes have defined subtraction differently. Our formal oscilloscope model exposes these distinctions to help the developer define a product that meets user requirements.

Another example that illustrates increased insight is the definition of a waveform, one of the first tasks in developing an oscilloscope model. We first asked software engineers who have built oscilloscopes to define waveform. Invariably, their answers reflected an implementation-level view, such as:

- A waveform is a 1-Kbyte array of eight-bit samples.

After some discussion it became clear that waveforms store time-voltage values, so we volunteered a second definition:

- A waveform is a function from time to volts.

Although this was a pleasing mathematical abstraction, we quickly discovered a serious flaw: Formally, this definition requires a voltage value for *every* time value. But when oscilloscope designers refer to a waveform, they are referring to *acquired* data, which represents voltage values over some time interval only. This realization led to an improved definition:

- A waveform is a partial function from time to volts.

This is much better (and is the definition we used in our simple model), but on closer inspection it is apparent that it cannot handle all waveforms that an oscilloscope can manipulate. In particular, some digital acquisitions store several values for each time value, such as the maximum and minimum value over some small interval centered on that time value. The third definition associates a unique voltage with each time value. This prompted a fourth, more general definition:

- A waveform is a relation between time and volts.

A relation is a collection of time-voltage pairs, so this definition encompasses the waveforms presented so far. But further discussion with the software engineers revealed that there are still other waveforms. In certain acquisition modes, waveforms are built out of repeated samplings. Many of these samples will be duplicates. For these waveforms, it is important to know how many repetitions there are of each time-voltage pair. As a set of pairs, our definition above precludes this. Thus we arrived at our final definition:

- A waveform is a bag of time-voltage pairs.

This example illustrates several things. First, it underscores the discrepancy between common practice in building software systems — in which designers view notions like waveforms at a detailed, implementation level — and the practice of modeling such entities in an abstract mathematical form.

Second, it illustrates how developing a good formal model is an interactive process that involves considerable domain-specific knowledge as well as mathematical expertise.

Third, it shows how precise mathematical notation helps analyze a definition's adequacy.

Fourth, it shows how the resulting abstraction both unifies and simplifies the ideas lurking behind a complex implementation. This in turn can lead to much cleaner implementations. For example, in the past, oscilloscope builders have had some difficulty managing the many varieties of waveforms that exist in an oscilloscope. In previous implementations, this was reflected in complex, special-purpose code for handling special waveform cases. The formal definition exposes what is common to all of these and leads to a much cleaner implementation.

While the goal of producing provably correct software is laudable, our use of formal reasoning has very different goals. We use mathematical reasoning to explore the properties of those specifications.

Formal reasoning. This nontraditional use of formal methods to gain insight into system design raises questions about the role of the formal proof. In most applications of formal methods, people usually use proofs to verify an implementation's correctness: Given a specification and a proposed implementation, a proof formally demonstrates that the implementation actually does what the specification says it should.

While the goal of producing provably correct software is a laudable one, our use of formal reasoning has very different goals. Instead of proving a formal relationship between specifications and their implementations, we use mathematical reasoning to explore the properties of those specifications.

In this oscilloscope work, our arguments were almost always informal, as the waveform example illustrates. Of course, if necessary, we could turn these informal

conjectures and arguments into formal theorems and proofs. Furthermore, we have found that if it is difficult to reason about some expected property, it is usually a sign that the specification is poorly structured, if not wrong.

In the oscilloscope model, for example, we might want to argue that traces portray input signals accurately. This is relatively easy to show for the simple oscilloscope. But if we tried to demonstrate that all trigger values result in some waveform acquisition, a formal argument would quickly reveal that our oscilloscope model is deficient.

A formal specification provides a precise, concise system description that forms a basis for formal and informal reasoning. The formal specification is both a document for communication and a tool that leads to increased insight and understanding of system behavior.

As a consequence, formal specifications take on a pivotal role in system development as *nonexecutable* prototypes. We use the term "nonexecutable" to contrast formal specifications with prototype code. In fact, a formal specification — such as the oscilloscope specification described here — may be executable. However, the benefits from executing such a specification are very different from those obtained by reasoning about it.

The insight gained from a formal specification has two tangible benefits. First, it can clarify user requirements, leading to products that better meet the users' needs. For example, we used the oscilloscope model to reason about the results of arithmetic operations on signals, waveforms, and traces. Second, it can form the basis for design. There are many possible designs that satisfy the formal specification presented here; there are also many factors not addressed here, such as details of the overall system and hardware architectures, that help determine which of these possibilities are the best. The formal model can play a key role in making complex problems intellectually manageable and in searching for solutions.

Clearly, there is much more work to be done to understand how formal specifications can best be used in industrial product development. However, our early experience suggests that they can play an important role in improving our understanding and engineering of complex software systems. ❖

Acknowledgments

Our work on applying formal methods to industrial products has been a collaborative effort. We gratefully acknowledge our many engineering partners in this endeavor. In particular, we thank Rich Austin, Don Birkley, Steve Lyford, David Olson, and John Wiegand for their help in opening to us the world of electronic instrumentation systems and for working with us to develop and promote formal frameworks. We also thank Ian Hayes, Kathleen Milstead, Ralph London, and Mayer Schwartz for their helpful comments on earlier drafts of this article. We are particularly indebted to Ian Hayes, who, through his extensions to our work, has given us great insight into what we were actually trying to accomplish.

References

1. G. Barrett, *Formal Methods Applied to a Floating-Point-Number System*, Tech. Report PRG-58, Programming Research Group, Oxford Univ. Computing Lab., Oxford, England, UK, 1987.
2. I. Hayes, *Specifying the CICS Application Programmer's Interface*, Tech. Report PRG-47, Programming Research Group, Oxford Univ. Computing Lab., Oxford, England, UK, 1985.
3. *VDM: The Way Ahead* (Proc. Second VDM-Europe Symp.), Lecture Notes in Computer Science 328, Springer-Verlag, New York, 1988.
4. C.B. Jones, *Systematic Software Development Using VDM*, Prentice-Hall, Englewood Cliffs, N.J., 1986.
5. D. Garlan and N. Delisle, "Formal Specifications as Reusable Frameworks," *VDM 90: VDM and Z!*, Springer-Verlag, New York, 1990, pp. 150-163.
6. I. Hayes, "Specifying Physical Limitations: A Case Study of an Oscilloscope," Tech. Report 167, Computer Science Dept., Univ. of Queensland, St. Lucia, Australia, 1990.

Norman Delisle is a principal computer scientist in the Computer Research Laboratory of Tektronix. His current research focuses on formal methods for software specification and design, object-oriented design, and software architectures for embedded systems. His earlier work was in specification tools, hypertext, programming environments, incremental compilation, and CASE tools.

Delisle received a BS in electrical engineering from Northwestern University and an MS in computer science from Worcester (Mass.) Polytechnic Institute.

David Garlan is a senior computer scientist in the Computer Research Lab of Tektronix. His research interests include the application of formal methods to the construction of reusable software architectures. He has been active in developing formal models of embedded instrumentation software and in building environments to support the development of these models.

Garlan received a PhD in computer science from Carnegie Mellon University, where his primary research was in the area of programming environments.

Address questions about this article to the authors at Computer Research Lab, Tektronix, PO Box 500, Beaverton, OR 97077.