Toward More Understandable User Interface Specifications

David A. Carr

Department of Computer Science and Electrical Engineering
University of Luleå
S-971 87 Luleå
Sweden
David.Carr@sm.luth.se

Abstract

Many different methods have been used to specify user interfaces: algebraic specification, grammars, task description languages, transition diagrams with and without extensions, rule-based systems, and by demonstration. However, none of these methods has been widely adopted. Current user interfaces are still built by writing a program, perhaps with the aid of a UIMS. There are two principal reasons for this. First, specification languages are difficult to use. Reading a specification and understanding its exact meaning is very difficult. Writing a correct specification is even more difficult. Second, most specification languages are not executable. This means that after the user interface programmer makes the effort to write a specification, the user interface must still be coded. As a consequence, most programmers have little incentive to do a specification. A pilot study into the comprehensibility of specifications is described. The results of this study suggest that user interface specifications are difficult to interpret manually. A possible solution to this problem, specification animation, is also described.

Keywords: specification comprehensibility, specification animation, user interface specification.

Introduction

Researchers have been formally specifying user interfaces for over fifteen years. However, very few systems outside of research labs have been formally specified. Common practice for user interface development is predominately based on programming. Most user interfaces are written with code and perhaps with the aid of a User Interface Management System that is little more than a layout editor. Almost all dialog is described only in the program. Whatever specification that is done is usually a text description, and it is rarely updated. Development organizations resist formal specifications because they are difficult to understand, are usually not executable, and therefore, are viewed as unnecessary additional work.

In order for specification to be widely adopted, the perceived benefits of specifying a system must outweigh the costs. To fulfill this goal specification systems must become more understandable and must be executable. An understandable specification system will allow user interface designs to be communicated between design team members and can serve as the basis for discussion about improvements and problems. An executable specification allows the development organization to skip the laborious and error prone processes of translating a specification into code. It should be noted that even an executable specification would require an understandable static representation. This representation would be used both for debugging and for communicating the design in situations where execution is not practical. A static representation can also help the designers visualize how different parts of the system relate to each other. Visualization is very difficult to achieve with a purely machine readable specification.

Of course, one must ask what "understandable" means. In this paper understandable is taken to mean that the specification reader can correctly determine the behavior of the user interface by reading the specification.

This paper first discusses various methods that have been used to specify user interfaces. It then describes a pilot study to compare the understandability of two different specification methods. Finally, it proposes specification animation as a way to improve specification understandability.

1 Previous Research

1.1 Interface Specification Methods

Over the years a number of methods have been used to specify user interfaces. These include algebraic specifications, grammar-based languages, transition diagrams (including extensions like Petri-nets and statecharts), rule-based systems, and specification by demonstration.

Algebraic specification of window systems was introduced by Guttag and Horning [8]. They proposed the design of a windowing system based on axiomatic specification of abstract data types. This method permits formally proving properties of the user interface. However, algebraic specifications have serious drawbacks. They are very difficult to read and require considerable time and training to understand. They are even more difficult to write. Specifically, the declarative nature of an algebraic specification provides little support for the sequential nature of user interface dialogs. This makes it difficult to construct or follow sequences of user actions, and therefore, makes algebraic specification unsuited for communicating interface behavior.

Systems that are grammar-based include Shneiderman's[27] multiparty grammars, Payne and Green's[22] task-action grammars, and Siochi, Hartson and Hix's User Action Notation (UAN) [10, 12]. Multiparty grammars were designed to model complete human-computer dialogs for command languages. A multiparty grammar divides non-terminals into three classes: user-input, computer, and mixed. User-input and computer non-terminals represent user actions and computer responses, respectively. Mixed non-terminals represent sequences in the human-computer dialogs. While reasonable models for command languages, multiparty grammars do not model the inherently non-sequential nature of direct manipulation well. Task-action grammars and UAN take a user-task point of view to specify command sequences. The operation of the system is organized into user tasks and the interface software is supposed to recognize sequences of user actions as a task. UAN includes support for specifying parallelism and other time dependencies[7, 11], and therefore, is better suited for specifying direct manipulation. The problem with the task-description strategy is that users don't always do exactly what the task description states. Thus, deriving a system from such a specification means that the translation tool must provide meaningful default error handling. This is not a simple job.

Another approach to modeling user interfaces is the transition diagram [32]. In this approach the transitions represent user inputs, and the nodes represent states of the interface. Computer outputs are specified as either annotations to the state or transitions. However, transition diagrams suffer from a combinatorial explosion in the number of states and transitions as system complexity increases. Jacob [16] solved part of this problem by allowing concurrent states to coexist as parallel machines or co-routines. Others have used extensions of the basic transition diagram. These extensions include statecharts and Petri-nets. Statecharts[13] were designed as a formal solution to the combinatorial problems with transition diagrams. The statechart adds the concept of a meta-state. Meta-states group together sets of states with common transitions that are inherited by all states enclosed in the meta-state. Since meta-states may enclose other meta-states, a complete inheritance hierarchy is supported. A special history state is supported to return the meta-state to its previous status on return transitions from events such as invoking help. Meta-states are divided into two types: parallel or AND-states and sequential or XOR-states. Meta-states enclosed within AND-states may execute in parallel and fulfill the function of co-routines. As originally defined, statecharts do not incorporate data flow or abstraction. Wellner[33] adapted statecharts to user interfaces by adding state entry and state exit actions to produce output from the specification. Rouff[26] used a statechart-like language called the Interface Representation Graph (IRG) as the underlying representation for his Rapid Program Prototyper. IRGs extend the statechart to represent dialog by introducing data flow, constraints, and communication with the application's back end. My own Interaction Object Graphs[2, 3] extend statecharts for widget specification by adding representations for display changes and widget attribute data. Another extension of the state diagram, the Petri-net, has also been used for user interface specification, for example Palanque and Bastide's Interactive Cooperative Objects[24]. The transition diagram and its extensions have two clear advantages as a formalism. First, they are easily translated into an executable form. Second, they have a graphical representation that can be easily animated to enhance user understanding of the system's execution.

Rule-based specification has been used in UIDE[5], PPS[21], and Sassafra[14]. All of these systems work in a manner similar to UIDE which uses pre- and post- conditions to control user interface dialog. User input causes one or more preconditions to become true. As preconditions become true associated actions are executed. These actions include post-conditions which may cause further changes in the interface state. Rule-based systems distribute the interactions between system components as interactions between different rules. This property makes them difficult to understand once they reach non-trivial size. It also makes them difficult to modify without introducing inconsistencies and unintended side-effects.

A final technique used for specifying human-computer dialog is to do so by demonstration. Using this technique the designer places widgets on the screen demonstrating state changes, and the system makes inferences about the intended design. "Druid" [30] and "Peridot" [20] are two systems which build user interfaces by demonstration. Druid provides for interactive layout and dialog definition. However, its dialog model does not include constraints between interface objects or the modification of interface objects. Peridot is a similar system which includes constraints and modification by redrawing. Neither of these systems has a clear visual representation for the user interface, although one could probably be developed. In fact, a state-based representation that was initially developed by demonstration and then further refined by an editor might be a very useful tool.

1.2 Animation

Animation has been used as an aid to understanding for data structures, programs, and algorithms. Systems which were built to illustrate changes in data structures include Incense[19] and VIPS[28]. The PECAN system[25] used multiple, graphical views of a program including both data and an animated flowchart. Visual programming systems such as Pict/D animated the program diagrams by highlighting the currently executing icon or flow path[6]. Pict/D programmers drew a flowchart of the program which was executed directly. Other systems such as VIPEX[9] are designed on a data flow paradigm. VIPEX nodes were connected via links at input and output pads. The programmer could selectively view data details at these pads while nodes would change color based on their status.

Other systems have concentrated on animating algorithms. With these systems the programmer constructs a program and a custom graphical representation of its abstract behavior. These systems include Balsa[1] and TANGO[31]. The Lens system[18] extends these systems by combining algorithm automation into a source-level debugger.

2 A Pilot Study on Understanding Specifications of Widget Designs

One of the most important uses of a specification is to communicate a design to others. However, how well a specification communicates that design is very difficult to measure. Yet, one would like to evaluate specification methods and determine which were better at communicating a design.

The specification methods with visual representations that are described above fall into two categories, diagrammatic and textual. The diagrammatic methods: transition diagrams, statecharts, Petri-nets, IRGs, and IOGs, are all based on some form of state machine. The textual methods are: rule-based specification, task-action grammars, multiparty grammars, and UAN. It would be useful to compare these two categories for understandability.

First, one must define "understandability". Since communication of the design to others is the goal, understandability should be defined in terms of the functions required to communicate the design. One measure of understanding is whether the reader can implement the widget. However, not all readers of the specification would be programmers. Other reasons to read the specification include: evaluating the design, documenting the widget, and verifying implementation correctness. A more useful measure of understandability would be performance in answering the following questions:

1. Given a sequence of user inputs: What is the status of important widget attributes?
2. Given a sequence of user inputs: What is the appearance of the widget on the display?
3. What sequence of user inputs is required to accomplish a given user task?
4. Given two similar specifications: What is the difference (if any) between them?

Answering the first two questions would be required for implementation, evaluation, documentation, and verification. The third question is important for evaluation and documentation. The last question would be important to someone evaluating successive designs of the same widget or trying to understand a modification to an existing widget.

The question remains whether there are any differences between text-based specifications and diagrammatic specification when communication is considered. A study comparing PASCAL syntax diagrams with BNF descriptions suggests that diagrams are superior for identifying syntax errors and for identifying the next legal language construct[4,17]. This study indicates that diagram-based specifications should be superior in answering the first two questions. Intuition would say that a specification method such as UAN should be superior in answering the third question. It is designed to give this type of information. The fourth question would probably be answered more quickly from a diagram, as long as the changes were more than the addition of an arc or two. However, it is quite possible to just rearrange the diagram and give the impression of change when none exists.

The rest of this section contains a brief description of a text specification method, UAN, a diagram specification method, IOGs, and a discussion of a pilot experiment comparing them. The experiment tries to measure differences in the number of correct answers and the time used on questions similar to the four above.

2.1 User Action Notation (UAN)

UAN is a text-based specification language for user interfaces that has its roots in user task description. A UAN specification is represented as a table divided into three columns: user actions, interface feedback, and interface state.

The first column is reserved for user actions. In this column the dialog designer writes a user action sequence in an event description language. The three most important symbols in this language are **Mv**, **M^**, and **~[Name]**. **Mv** and **M^** represent pressing and releasing the first mouse button, respectively. **~[Name]** represents moving the mouse into the context of the object **Name**. User action sequences may be prefaced with a precondition which is enclosed in parenthesis. A precondition must be true before the user action sequence will have any effect. For example, the user action, “**(Sw=OFF): Mv**”, means that the interface state variable **Sw** must have the value **OFF** before any interface changes beginning with the user pressing the mouse button will apply.

The second column in a UAN table describes the interface feedback. This is done with a text description of the changes such as “**display(ActiveHeat)**”. This states that the image named **ActiveHeat** will be shown on the display. There are other feedback operators described in the references[10, 11, 12].

The third column in a UAN table is labeled interface state. This column specifies changes in state variables that are used in the preconditions described above. The variables represent state information about the user interface or the application. An example of this is “**Sw = OFF**” which assigns the value **OFF** to the variable **Sw**. UAN state variables may be of any type.

UAN specifications are read by starting

Task: Operate = FirstDisplay (HeatOff | OffCool | CoolOff | OffHeat)*

Task: HeatOff		
User Action	Interface Feedback	Interface State
(Sw = HEAT): ~[Heat]Mv	display(ActiveHeat)	
((~[Off2Heat]	display(Heat-Off)	
~[Off]	display(ActiveOff)	Sw := OFF
M^)	display(Off)	
(~[Off2Heat]	display(Heat-Off)	
M^) M^)	display(Heating)	

Task: CoolOff		
User Action	Interface Feedback	Interface State
(Sw = COOL): ~[Cool]Mv	display(ActiveCool)	
((~[Off2Cool]	display(Off-Cool)	
~[Off]	display(ActiveOff)	Sw := OFF
M^)	display(Off)	
(~[Off2Cool]	display(Off-Cool)	
M^) M^)	display(Cooling)	

Task: OffCool		
User Action	Interface Feedback	Interface State
(Sw = OFF): ~[Off]Mv	display(ActiveOff)	
((~[Off2Cool]	display(Off-Cool)	
~[Cool]	display(ActiveCool)	Sw := COOL
M^)	display(Cooling)	
(~[Off2Cool]	display(Off-Cool)	
M^) M^)	display(Off)	

Task: OffHeat		
User Action	Interface Feedback	Interface State
(Sw = OFF): ~[Off]Mv	display(ActiveOff)	
((~[Off2Heat]	display(Heat-Off)	
~[Heat]	display(ActiveHeat)	Sw := HEAT
M^)	display(Heating)	
(~[Off2Heat]	display(Heat-Off)	
M^) M^)	display(Off)	

Task: FirstDisplay		
User Action	Interface Feedback	Interface State
(Sw = OFF):	display(Off)	
(Sw = HEAT):	display(Heating)	
(Sw = COOL):	display(Cooling)	

Figure 1 -- UAN specification of a common thermostat.

Figure 2 -- IOG node symbols.

in the top-left and proceeding row-by-row from left-to-right. Complex actions are specified by composing simple tasks. For example, the **Operate** task in Figure 1 which specifies a heating control switch that is common in the U. S. This switch was used as part of the pilot study described below. In addition to the UAN, pictures representing the displays were provided.

2.2 Interaction Object Graphs

Interaction Object Graphs (IOGs) are designed to add widget specification to Interface Representation Graphs. They combine the data flow and constraint specifications of IRGs with the statechart, transition-diagram, execution model. This expands the statechart to show data relationships as well as control flow. IOGs add a display state and a representation for widget attribute data in order to permit specification of low-level interaction objects which cannot be specified by Interface Representation Graphs. Below is a brief description of the IOG state diagram, and a transition description language used to specify transition conditions.

The IOG state diagram traces its lineage from UAN, statecharts, and IRGs. Statecharts added four new state types to the traditional state diagram. These states are used in IOGs. They are: the XOR meta-state, the AND meta-state, and two types of history state.

The meta-states can contain both normal states and other meta-states. Transitions from meta-states are inherited by all contained states. This helps reduce the problem of arc explosion. The XOR meta-state contains a sequential transition network. Exactly one state inside of an XOR meta-state is active when the XOR state is active. On the other hand, an AND meta-state contains more than one transition network. Each of these networks executes in parallel.

A history state can only be contained in an XOR meta-state. Whenever a transition transfers control from a meta-state, the history state remembers which state was active immediately before the transition. If a later transition returns control to the history state, the meta-state is returned to the remembered state. History states help control state explosion. To see this, consider a specification of a help system which is independent of the user interface. An ordinary transition network would require replicating the help-system specification once for every state in the user interface. Otherwise, there would be no way to return to the user interface state that was active before help was requested. A statechart history state could receive the return transition from the help system, and only one copy would be required. There are two types of history states. They differ in how they treat a return when the last active state was a meta-state. The **H** state restarts meta-states at their start state and provides one level of history. On the other hand, the **H*** state restarts meta-states at their history state, when they have one, thereby allowing multilevel history. Figure 2 shows the representation of the new states.

IOGs add two additional node types to the statechart, data objects and display states. Data objects represent the storage of a data item, and control is never passed to them. They can only be destinations for the data arcs discussed below. Display states are control states that have a change in the display associated with them. In IOG diagrams a picture of the display change is used whenever possible instead of a program-like statement such as "draw(ActiveON)". Data objects are represented as parallelograms. (Figure 2.)

IOGs also add two special arc types, the event arc and the data arc. Events allow the designer to define "messages" which may be lacking in the underlying specification model. For example when specifying the trash can in the

Figure 3 -- IOG arc symbols

Macintosh interface, one needs to know when a file is being dragged over it as opposed to when the pointer is being dragged over it. One way to do this would be for the file icon to generate a "dragging started" event and a "dropped" event. The trash can would then be highlighted whenever the pointer was over it between a "dragging started" event and a "dropped" event. An event is represented by a special transition passing through an **E** in a diamond. (Figure 3).

Data flow is represented in a manner similar to events – an arc passing through a **D** in a circle (Figure 3). A data flow arc may have any state as a source and can only terminate at a data object or have an unspecified termination. In addition, at least one end must be attached to a data object. Data flow arcs with data objects as a source, whose destination arrow is unspecified, and whose destination is outside of the containing interaction object, indicate externally-readable data (Figure 4). This data may be used by the application or attached to other user interface components as a more complete specification is constructed. Data flow arcs with data objects as destinations represent updating the data object. If the arc's source is a control state, it represents a change in the value when the arc conditional is satisfied. In this case, the data flow arc is labeled with the new object value. An arc without a source represents externally-writable data.

Constraints are useful in specifying one attribute of the user interface in terms of others. With constraints it is simple to restrict an icon to be contained within a window or to map the values of a slider to a specific range. IOG data arcs support a form of one-way constraints by expressing the data value as an equation in terms of other attributes. Together with the condition on the data arc, these equations provide a means to constrain one attribute in terms of another with a Boolean guard. For example, specifying that in a given state (call it **S**), the image of a widget will follow the mouse cursor can be done with a constraint. This is accomplished by drawing a data arc from **S** to the data object representing the widget's location. Now, specify that the new value for the location is: "old location + the change in the mouse position". The condition for the arc would be: "the mouse position changes". This results in the widget following the mouse while the widget control is in state **S**.

In order to describe the transitions between states, an abstract model of the user interface and a description language for that model are required. IOGs abstract the interface into the following objects: Booleans, numbers, strings, points, regions, icons, view ports, windows, and user inputs. A brief description of these objects follows.

Booleans, numbers, and strings (BNS) are the usual abstractions with the usual operations. It should be noted that numbers contain both the real and integer data types. In addition, any of these may be converted into an icon representation by the operator **icon(BNS, point, font, fontsize)** or **icon(BNS, region, font)**. Both operators convert the Boolean, number, or string **BNS** to a text representation, and then convert the text representation into a picture. If specified with a point, the resulting icon is as big as it needs to be to hold the picture of the text representation. If a region is specified, the icon is the size of the region. Fonts and font sizes may be omitted. In that case defaults will be used.

Points are an ordered pair of numbers (x,y). Points have the algebraic operators which are normally associated with them. A point may be assigned a value by writing **p=(x,y)**. In addition, **p.x** and **p.y** represent the x and y coordinates from the point **p**.

A region is a set of display points defined relative to an origin called the **location**. The location of the region is always the point (minx,miny) where minx and miny are the smallest x and y coordinates in the region. Regions have a **size** operator which returns the height and width of the smallest rectangle which covers the region. Regions also have an **in** operator which tests if a point is in the region. This is written **Region.in(pt)** and returns a Boolean value.

Figure 4 -- A readable and writable widget attribute in an IOG.

Although regions are not restricted to be rectangular, rectangles are most commonly used. Note, a region cannot be visible on the display. There is no drawing operation associated with a region.

Icons are regions with pictures. That is, some points in the region have a color number attached to them and are shown on the display. Icons add the operations **draw** and **erase**. In addition, if the origin of the icon is changed, there is an implicit **erase-draw** operation sequence. Unless otherwise specified, the region associated with an icon is a rectangle. So, `icon("text", pt, default, default)` would product an icon with the upper-left corner at **pt**. The picture would be the word "text" in the default font and size. There would be a region associated with the icon. This region would be a rectangle with its upper-left corner at **pt** and of sufficient area to cover the text.

A view port is a region with an associated mapping function for some underlying application data. The mapping would be in two parts, conversion to a world-coordinate-system graphics representation and projection onto the display. For example, text would first be converted from ASCII to a font representation and a location on a page. The page would then be projected onto the display. The mapping is controlled by a projection function (**proj**), a translation point (**translate**), and a scale-change point (**scale**). If **convert** is the conversion function for some object in some view port, then the function **translate + proj(scale, (convert(object)))** would be the view port mapping. Parts of objects projected to points not in the region are not displayed, and objects in view ports are addressed relative to the view port location.

Windows group the above objects together. They add a level attribute which determines window stacking relative to other windows. They can be viewed as view ports containing only objects already mapped to display coordinates. A window assigned a lower-level value obscures an overlapping window assigned a higher-level value.

Objects are addressed in the specification using a dot notation. For example, "win.icon1.location.x" would be the x coordinate of the location of icon, "icon1", in window, "win".

User inputs are mapped to IOG events, numbers, points, and Boolean variables. Keyboard input is represented by quoted strings when the text is important ("quit-") when the word "quit" is typed and followed by a carriage return) or key events, similar to those in UAN, when the event is important (**LShiftv** for left shift key pressed). The mouse is mapped into a point for location (**M@**), a point for relative change (**MΔ**), a Boolean indicating it moved (**ΔM**), button change events (**Mv**, **M^**, **M2v**, ...), and button status variables (**Mdn**, **Mup**, **M2dn**, ...). Since the value of the mouse location is tested frequently, **in[Region]** is written as a shorthand for **Region.in(M@)**. The special notations **~[Region]** and **[Region]~** describe the events of the mouse entering and leaving the **Region**. These symbols can be combined in expressions. The operators from the 'C' language are used for these expressions. (Most commonly, "&&" for logical AND, "||" for logical OR, and "!" for logical NOT.)

Figure 5 shows the IOG specification for the same heating-control switch as was shown for UAN in the previous section.

2.3 Pilot Experiment Description

The experiment utilized a between-groups design. There were two groups of participants. All participants were graduate students from a graduate class in programming languages and software engineering. A large segment of this class is devoted to software specification methods. Participants were given a reading assignment of a conference paper[29] on UAN and a conference paper[2] on IOGs as homework prior to the day of the experiment. All participants simultaneously listened to a forty-five minute lecture on UAN and IOGs. Other than this lecture the experimenter had no contact with the class. During the lecture a secure-switch specification (Figure 6) was presented for both methods, and the participants were given a tracing task as practice for both UAN and IOGs. The participants could ask questions at any time. After the lecture, participants had the option of declining to participate in the

Figure 5 -- IOG specification of a common thermostat.

IOG

Subject ID	A1	A2	A3	A4	A5	A6	Total
Question #1	1	0	1	1	1	1	5
Question #2	1	0	1	0	0	0	2
Question #3.1	1	0	1	1	0	1	4
Question #3.2	1	1	1	0	0	1	4
Question #3.3	1	0	1	0	0	1	3
Question #3.4	1	0	1	0	0	1	3
Question #3.5	0	0	0	0	0	0	0
Total Correct	6	1	6	2	1	5	21

UAN

Subject ID	B1	B2	B3	B4	B5	B6	Total
Question #1	1	1	1	1	1	0	5
Question #2	1	0	0	0	0	0	1
Question #3.1	0	1	1	1	1	0	4
Question #3.2	0	1	1	1	1	0	4
Question #3.3	0	1	0	0	0	0	1
Question #3.4	0	0	0	0	0	0	0
Question #3.5	0	1	0	0	0	0	1
Total Correct	2	5	3	3	3	0	16

Table 1 -- Individual and total number correct for Part 1. (1 = Correct).

remainder of the experiment. Those that elected to remain were randomly divided into two groups, A and B. The experiment was divided into four parts. Group A used IOGs in Part 1, UAN in Part 2, and IOGs in Part 3. Group B used UAN, IOGs, and UAN for Parts 1-3, respectively. Part 4 asked for a subjective evaluation of the two specification methods.

Each group received a questionnaire which presented a specification and then a series of questions about that specification. The questions and widgets were identical for each group, only the specification changed. The independent variable was specification type. Number of correct responses and time to complete were measured as dependent variables. Time values were displayed on an overhead projection of a computer-generated screen with elapsed minutes and seconds. Participants were asked to write the time before and after each question in the questionnaire. The questionnaires that the participants received were copied on two sides. When the questionnaire was opened like a book, the left-hand side always contained a copy of the current specification. The right-hand side had the description of the user actions and the questions. There was a page with the message "PLEASE, STOP HERE AND WAIT FOR INSTRUCTIONS TO CONTINUE" after both Parts 1 and 2. (Due to space limitations only the specifications for Part 2 are presented here as Figures 1 and 5. All specifications can be found in [3].)

Part 1 asked participants to trace through a specification and to determine the system's response. They were given an initial state and a sequence of user actions and asked questions about the final state of the interface. The widget specified in Part 1 was a file icon similar to that used in the Macintosh desktop. However, the widget specification contained an error that gave the file icon a different dragging behavior than a Macintosh file icon. Participants were warned that an error existed, and that they should trace specification behavior, not expected behavior. Participants were given ten minutes to study the specification and answer seven questions. The participants were presented with a sequence of user actions followed by one or more questions. There were three user-action sequences with one, one, and five questions. Because the experiment was given in a seventy-five minute class period, time for Part 1 was limited to 10 minutes.

Part 2 also asked participants to trace through a specification and to determine the system's response. The specification in Part 2 was of a three-position, heating/cooling switch similar to the Heat-Off-Cool switch found on many residential thermostats. As in Part 1, there were three user-action sequences but with two, three, and two questions. Time for Part 2 was also limited to ten minutes.

Part 3 specified a modified secure switch with added feedback when the user moved outside of all switch regions just before releasing the mouse. Two questions were asked in Part 3. The first asked, "What has changed?" The second question asked the users to just write down the sequence of user actions required to turn the switch from "off" to "on". The first question tests if the visual nature of IOGs helps participants remember previous specifications and spot modifications of those specifications. The second question might be easier using UAN as participants must simply write the sequence from the user task description while with IOGs they must trace a series of arcs.

Part 4 asked each participant two questions of the form: "Rate the XXX specification method for understanding and ease of use." Where "XXX" was IOG and UAN, and the rating was on a scale of one-to-nine with nine as best. Participants had unlimited time for Parts 3 and 4.

2.4 Results

Twelve graduate students elected to participate in the experiment. They were randomly assigned into group A or group B with six in each group.

In Part 1, the IOG group (A) got a higher number of correct answers (21 vs. 16). (See Table 1). However, the result was not statistically significant using a t-test ($t = .504$). In addition to the t-test, Fisher's exact tests were computed to see if any individual questions had significant differences for the number correct between the IOG and UAN groups, and none were found at the $\alpha=.05$ level. What is interesting to note is that the error rates were very high, 50% for IOGs and 62% for UAN. When questions that were not answered are not counted, the error rates still remain high at 36% for both groups combined.

For Part 2, the IOG group (B) again got more correct than the UAN group (36 vs. 33). (See Table 2). However, the result was not statistically significant using a t-test. As was done for Part 1, Fisher's exact tests were computed to see if any individual questions had significant differences for the number correct between the IOG and UAN groups. In Part 2, there were no questions with significant differences between specification methods. Part 2 was an easier question, and the participants had more practice, but the error rates were still high at 16% for IOGs and 22% for UAN. When unanswered questions are ignored, the error rate still remains at 18% for both groups combined.

<i>IOG</i>								<i>UAN</i>							
Subject ID	B1	B2	B3	B4	B5	B6	Total	Subject ID	A1	A2	A3	A4	A5	A6	Total
Question #1.1	1	1	1	1	1	0	5	Question #1.1	1	1	1	1	1	1	6
Question #1.2	0	1	1	1	1	1	5	Question #1.2	0	0	1	1	1	1	4
Question #2.1	1	1	1	1	1	1	6	Question #2.1	1	1	1	1	0	1	5
Question #2.2	1	1	1	1	1	1	6	Question #2.2	1	1	1	1	1	1	6
Question #2.3	1	1	1	1	1	1	6	Question #2.3	1	1	1	1	0	1	5
Question #3.1	1	1	1	1	1	1	6	Question #3.1	0	1	1	1	0	1	4
Question #3.2	1	1	0	0	0	0	2	Question #3.2	0	1	1	0	0	1	3
Total Correct	6	7	6	6	6	5	36	Total Correct	4	6	7	6	3	7	33

Table 2 -- Individual and total number correct for Part 2. (1 = Correct).

Part 3 asked two questions designed to test different aspects of the difference between diagrams and linear text specifications. The first question presented a modified version of the secure switch and asked the participants what had changed. The participants were shown both UAN and IOG specifications of the secure switch during the lecture. It was expected that visual memory of the IOG specification would result in more IOG participants being able to describe the changes made to the IOG specification. Although the four IOG users were correct as compared with only one UAN user, Fisher's exact test did not indicate that this difference was significant (Table 3).

The second question asked participants to write the sequence of user actions required to turn the switch from off to on. Since UAN was designed to specify user tasks, it was expected that UAN would be better than IOGs for this question. However, there was no significant difference between methods for this question. (See Table 3.)

<i>Recognize Changes</i>			<i>List User-Action Sequence</i>		
	Right	Wrong		Right	Wrong
IOG	4	2	IOG	5	1
UAN	1	5	UAN	4	2
Fisher's exact test P(UAN = IOG)	0.12		Fisher's exact test P(UAN = IOG)	0.50	

Table 3 -- Fisher exact test for Part 3.

The final part of the questionnaire asked participants to rate UAN and IOGs for understandability and ease of use. Question one was, "Rate the IOG specification method for understanding and ease of use." Question two simply substituted UAN for IOG. A scale of one-to-nine was used. Instead of analyzing the raw scores, a difference was computed by subtracting the UAN rating from the IOG rating (Table 4). This difference score gives a relative, subjective rating of the participants' opinion of the differences between UAN and IOGs.

The mean difference was computed for both groups, and a t-test was performed to test the hypothesis that both groups had the same mean. The mean differences were not the same for each group (t-test, $P(\text{IOG}=\text{UAN}) = .015$). Therefore, a t-test was made for each group to see if the difference was significant (Table 5). For group A, no significant difference was found. However, a significant difference was found ($p = .029$) for group B.

It is possible that there was some undetected difference between groups. However, the participants were randomly assigned from a unusually homogeneous population. All were graduate students in the same first-level graduate course, had been given the same reading assignments, and had exactly the same training lecture. Therefore, the most likely reason for this is that Group B had UAN for both the "file icon" questions and the "what's different" questions. These would have been more difficult with UAN, and this probably lead the group B participants to rank IOGs higher. In particular, the "what's different" question would have pointed out the difficulty in spotting modifications in UAN's tabular format. They also used UAN first and would have had poorer scores with UAN because of inexperience.

2.5 Conclusions

The pilot experiment gives some support to the theory that diagrams are better for understanding specifications. The software-engineering students did better with IOGs than with UAN. Although no statistically significant results were found for correctness, participants with IOGs got the correct interpretation more often. Those participants who used more UAN specifications preferred IOGs.

The pilot experiment might have had more significant results if there had been more participants. With only twelve participants, statistical significance is hard to obtain for any experiment involving humans. In addition, a longer questionnaire and more time would have been helpful. More questions would have given participants more experience. Removing time pressure might have lead to lower error rates and shown significant differences in time to complete. However, more time and more participants were not available.

Perhaps the best way to compare specification methods would be to have competing teams of equal skill developing the same widgets using different specification methods. This would be a natural next step as part of a more extensive experiment. Another useful test would be field trials with professional programmers working on commercial products.

The experiment did point out the difficulty in tracing specifications by hand. For questions that they answered, participants had error rates of 36% and 18%, for parts 1 and 2, respectively. Clearly, computer support for understanding specifications would be beneficial.

3 Animating Specifications

The pilot experiment in the previous section illustrates the difficulty in tracing a specification. Error rates were between 18% and 36%. In addition, my own personal experience indicates that it is very difficult to explain a specification to others. Experience with specifications also shows that it is rather difficult to specify "good" interactions. That is, a specification that looks good on paper turns out to be rather difficult to use.

These problems motivated a search for a way to make specifications easier to understand and test. Since IOGs are executable and have a graphical representation, it seemed natural to try animating a IOG specification diagram while simultaneously executing the widget. With an animated specification the readers can test the specification and improve their understanding of its meaning. In addition, the widget designer could directly observe specification behavior while designing and debugging the specification.

If we consider this animation design with respect to the four usability questions below:

1. Given a sequence of user inputs: What is the status of important widget attributes?
2. Given a sequence of user inputs: What is the appearance of the widget on the display?
3. What sequence of user inputs is required to accomplish a given user task?
4. Given two similar specifications: What is the difference (if any) between them?

We can see that the first two questions can be answered directly by running the animation on the user-input sequence.

Subject ID	A1	A2	A3	A4	A5	A6	B1	B2	B3	B4	B5	B6
IOG Rating	3	7	7	6	3	7	8	7	4	7	7	9
UAN Rating	2	6	3	6	5	6	5	3	1	5	2	4
Difference	1	1	4	0	-2	1	3	4	3	2	5	5

Table 4 -- Subjective ratings.

	Group A	Group B
Number of Subjects	6	6
Mean Difference	0.83	3.67
Standard Deviation	1.95	1.21
t-score	0.43	3.03
P(IOG=UAN)	.686	.029

Table 5 -- Subjective ratings statistics.

Animation is not as much help answering the third question. It can be used to verify a sequence will accomplish the desired result, but it cannot be used to guarantee that it is the shortest. If many questions of this kind need to be answered, then a path search on IOG transitions would be better. For the final question, animation allows the reader to test the differences between two specifications by applying the same sequences of input to each, but again the reader cannot obtain a direct answer.

3.1 Design of IOG Animation

The first problem when animating IOGs is to decide what to display. Clearly, one would want to display the widget which has been specified. One also needs a IOG diagram. IOGs have two main components, arcs and nodes. One possible animation would be to highlight each node which is active and highlight each arc as it activates. For example, each active state could be outlined in red. As the IOG interpreter executes, arcs attached to each active state are tested. When one of these arcs has a true precondition, that arc could then be changed to red. After all arcs have been tested, the IOG interpreter updates the list of active states. Each newly active state could be outlined while the outline was removed from the deactivated states. At this time, the arcs would be returned to their normal appearance.

However, there are some efficiency and aesthetic problems with the above approach. The bottleneck in any user interface system is updating the display. This is especially true of the IBM PC clone used to display the animation. As more arcs are activated, the operation of the widget lags farther behind the user. This results in very poor operation of the widget. In addition, there are problems with data nodes. They are not really ever active. They could be flashed as they update, but this affects the system response when objects are being dragged. Also, there is the question of outlining meta-states. For the secure switch, this would result in three meta-states being permanently outlined: the entire widget diagram (an AND meta-state), the area of the diagram not containing the data objects, (a XOR meta-state), and either "Static States" or "Operating States". This seems excessive. Rather than impair the performance of the widget itself, a more limited animation was used. Only active, atomic states were outlined for the test system. This seems to give a good idea of how the widget operates without cluttering the display with outlines and without slowing widget execution.

IOGs are an executable specification method and a library of C++ classes modeling the IOG abstractions had been implemented. A program that used these classes to demonstrate a few widgets was augmented to produce the animated versions of these widgets.

3.2 IOG Animation Example

This section gives a brief description of how the animation will appear. The widget shown is a secure toggle switch based on a design by Plaisant[23]. The switch was designed for operation on a touch screen control panel and is secure in the sense that an accidental tap on the switch cannot cause a state change. In order to cause the switch to change the user must press on the side representing the current state of the switch, drag the switch through a middle position into the side representing the new state, and finally, lift-off from the touch screen while pointing at a region corresponding to the new state. Lifting-off from the touch screen in any other region of the screen cancels the operation and returns the switch to its original position.

Figure 6 depicts the animation as the user turns the secure switch from off to on. Before the user begins to manipulate the switch, the widget and the diagram appear on the display. Figure 6a shows this state with the mouse pointing in the **Off** region. The upper-right display state is outlined in red on the display. (This appears as a wide rectangle in Figure 6a.) When the user presses the mouse button while pointing in the **Off** region, the animated diagram changes. The lower-right display state is now outlined (Figure 6b). (Note, the mouse cursor has changed to be the outline of a mouse. This change is used when demonstrating for large audiences to inform them that the mouse button has been pressed. Changing the cursor makes an invisible user action visible.) Next, as the user moves the mouse into the **Mid** region with the mouse button pressed, the IOG diagram changes so that the lower-middle display state is outlined (Figure 6c). As the user continues moving the mouse into the **On** region without releasing the mouse button, the IOG diagram changes with the lower-left display state outlined (Figure 6d). Finally, when the user releases the mouse button, the upper-left state is outlined (Figure 6e).

While no study has been done to evaluate the effectiveness of animation in improving specification understanding, animation has made it much easier to explain the IOG specification method. Animation viewers grasp the essentials of IOGs quicker. Also, they begin to find usability problems with the widgets specified and can propose solutions. (For example, the **On** and **Off** regions are not explicitly displayed for the user, and there is no state explicitly indicating that the switch will revert to its original value. This can be fixed by adding two more display states showing that the switch will revert.)

a) After user input, $\sim[Off]$.

b) After user input, Mv

c) After user input, $\sim[Mid]$.

d) After user input, $\sim[On]$.

e) After user input, $M^$.

Figure 6 -- Animation of the secure switch.

Conclusion

Current specification methods are difficult to understand. In addition, many are not executable. If specification is to become an accepted practice in the software industry, both problems must be overcome. Preliminary work suggests that a possible solution to these problems would be an executable specification with a diagrammatic representation that could be animated as the specification was executed. However, more research is required. An experiment to determine if animation actually helps is necessary. If this proves to be the case, much work is required to scale the animation and specification method to larger interfaces. The class library itself needs to incorporate the animation more directly, and better tools for browsing and editing the specification are required. In any case I believe that the animation technique could be applied to any specification based on a transition diagram or one of its extensions.

Acknowledgments

I wish to thank Carl Rollo for his work in implementing the Microsoft Windows' interface and implementing the animated specification. I would also like to thank him for proofreading drafts of this paper. Special thanks also go to Sylvia Sheppard and Christopher Rouff of NASA Goddard Space Flight Center for their encouragement and support. Much of this work was carried out while I was at the Human-Computer Interaction Laboratory of the University of Maryland, and I would like to thank my colleagues there for their suggestions for improvements in the development of IOGs.

References

1. Brown, M. H. and R. Sedgewick, "Techniques for Algorithm Animation", *IEEE Software*, Vol. 2, No. 1, Jan., 1985, pp. 28-39.
2. Carr, David, "Specification of Interface Interaction Objects", *Proceedings of ACM CHI'94 Conference on Human Factors in Computing Systems*, 1994, pp. 372-378.
3. Carr, David, *A Compact Graphical Representation of User Interface Interaction Objects*, University of Maryland, Department of Computer Science, Ph.D., 1995.
4. Fitter, M. and T. R. G. Green, "When Do Diagrams Make Good Computer Languages?", *International Journal of Man-Machine Studies*, Vol. 11, No. 2, 1979, pp. 235-261.
5. Gieskens, Daniel and James Foley, "Controlling User Interface Objects Through Pre- and Postconditions," *Proceedings of the ACM CHI'92 Conference on Human Factors in Computing Systems*, 1992, pp. 189-194.
6. Glinert, Ephraim P. and Steven L. Tanimoto, "PICT: An Interactive Graphical Programming Environment", in *Visual Programming Environments: Paradigms and Systems*, Ephraim P. Glinert, editor, IEEE Computer Society Press, Los Alamitos, CA, 1990, pp. 265-283.
7. Gray, P. D., D. England, and S. McGowan, "XUAN: Enhancing UAN To Capture Temporal Relationships Among Actions", in G. Cockton, S. W. Draper and G.R.S. Weir, ed., *People and Computers IX*, 1994, Cambridge University Press, Cambridge, UK., pp. 301-312
8. Guttag, John and J. J. Horning, "Formal Specification as a Design Tool," *Proceedings of the Seventh ACM Symposium on Principles of Programming Languages*, 1980, pp. 251-261.
9. Haarslev, Volker and Ralf Möller, "VIPEX: Visual Programming of Experimental Systems," in *Visual Languages and Visual Programming*, Shi-Kuo Chang, editor, Plenum Press, 1990, New York, NY, pp. 185-212.
10. Hartson, H. Rex, Antonio C. Siochi, and Deborah Hix, "The UAN: A User-Oriented Representation for Direct Manipulation Interface Designs", *ACM Transactions on Information Systems*, Vol. 8, No. 3, July, 1990, pp. 181-203.
11. Hartson, H. Rex and Phillip D. Gray, "Temporal Aspects of Tasks in the User Action Notation", *Human-Computer Interaction*, 1992, Vol. 7, Lawrence Erlbaum Associates, pp. 1-45.
12. Hartson, H. Rex and Kevin A. Mayo, "A Framework for Precise Reusable Task Abstractions", *Proceedings of the Eurographics Workshop on Design, Specification, and Verification of Interactive Systems*, Fabio Paterno, editor, 1994, pp. 147-164.

13. Harel, David, "On Visual Formalisms", *Communications of the ACM*, Vol. 31, No. 5, May, 1988, pp. 514-530.
14. Hill, Ralph D., "Supporting Concurrency, Communication, and Synchronization in Human-Computer Interaction - The Sassafras UIMS," *ACM Transactions on Graphics*, Vol. 5, No. 3, July, 1986, pp. 179-210.
15. Hudson, Scott E., "Graphical Specification of Flexible User Interface Displays," *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology*, 1989, pp. 105-114.
16. Jacob, Robert J. K., "A Specification Language for Direct-Manipulation User Interfaces," *ACM Transactions on Graphics*, Vol. 5, No. 4, October, 1986, pp. 283-317.
17. Jensen, Kathleen and Nicolas Wirth, *PASCAL User Manual and Report*, Springer-Verlag, New York, 1975.
18. Murkherjea, Sougata and John T. Stasko, "Toward Visual Debugging: Integrating Algorithm Animation Capabilities within a Source-Level Debugger," *ACM Transactions on Computer-Human Interaction*, Vol. 1, No. 3, Sept. 1994, pp. 215-244.
19. Myers, Brad A., "A System for Displaying Data Structures", *Computer Graphics*, Vol. 17, No. 3, July, 1983, pp. 115-125.
20. Myers, Brad A., *Creating User Interfaces by Demonstration*, Academic Press, Boston, 1988.
21. Olsen, Dan R. Jr. and Kirk Allan, "Creating Interactive Techniques by Symbolically Solving Geometric Constraints", *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology*, 1990, pp. 102-107.
22. Payne, Stephen J and T. R. G. Green, "The Structure of Command Languages: An Experiment on Task-Action Grammar", *International Journal of Man-Machine Studies*, Vol. 30, No. 2, Feb., 1989, pp. 213-234.
23. Plaisant, Catherine, and D. Wallace, "Touchscreen Toggle Design", *Proceedings of the ACM CHI'92 Conference on Human Factors in Computing Systems*, 1992, pp. 667-668 (Video).
24. Palanque, Phillipe, and Rémi Bastide, "Petri Net Based Design of User-Driven Interfaces Using the Interactive Cooperative Objects Formalism", *Design, Specification and Verification of Interactive Systems '94 Proceedings*, Fabio Paterno', ed., 1994, pp. 215-228.
25. Reiss, Steven P., "PECAN: Program Development Systems that Support Multiple Views", in *Visual Programming Environments: Paradigms and Systems*, Ephraim P. Glinert, editor, IEEE Computer Society Press, Los Alamitos, CA, 1990, pp. 324-333.
26. Rouff, C. and E. Horowitz, "A System for Specifying and Rapidly Prototyping User Interfaces", *Taking Software Design Seriously*, John Karat, ed., Academic Press, 1991, pp. 257-272.
27. Shneiderman, Ben, "Multiparty Grammars and Related Features for Defining Interactive Systems," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 12, No. 2, 1982, pp. 148-154.
28. Shimomura, T. and S. Isoda, "Linked-List Visualization Systems", *IEEE Software*, Vol. 8, No. 3, May, 1990, pp. 44-51.
29. Siochi, Antonio C., and H. Rex Hartson, "Task Oriented Representation of Asynchronous User Interfaces", *Proceedings of the ACM CHI'89 Conference on Human Factors in Computer Systems*, 1989, pp. 325-330.
30. Singh, Gurminder, Chun Hong Kok, and Teng Ye Ngan, "Druid: A System for Demonstrational Rapid User Interface Development", *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology*, 1990, pp. 167-177.
31. Stasko, John T., "TANGO: A Framework and System for Algorithm Animation", *Computer*, Vol. 23, No. 9, Sept., 1990, pp. 27-39.
32. Wasserman, Anthony I., "Extending State Transition Diagrams for the Specification of Human-Computer Interaction", *IEEE Transactions on Software Engineering*, Vol. SE-11, No. 8, August, 1985, pp. 699-713.

33. Wellner, Pierre D., "Statemaster: A UIMS Based on Statecharts for Prototyping and Target Implementation Notation for Specification," *Proceedings of the ACM CHI'89 Conference on Human Factors in Computing Systems*, 1989, pp. 177-182.