STATE PROBLEMS IN PROGRAMMING HUMAN-CONTROLLED DEVICES

Joseph A. Konstan
Department of Computer Science
University of Minnesota
Minneapolis, MN 55455

ABSTRACT

Many consumer goods are complicated enough to benefit from programmed control. Today's home electronics devices support a wide range of options and controls. At the same time, personal digital assistants and programmable remote controls are now capable of learning and generating control sequences to control a wide range of devices. Unfortunately, most device interfaces are designed for interactive human control rather than programmed control.

This paper analyzes state-based obstacles to programming devices designed for interactive human control. It develops a theory of statelock, a condition in which a control program is unable to synchronize with the state machine underlying the controlled device. The paper also presents design strategies to avoid statelock and applies these strategies to the home audio/video and telephone autodialer domains.

KEYWORDS: Device interface, programmable remote control, automata, user/machine systems, audio/video control, telephone autodialers.

INTRODUCTION

Many consumer goods are complicated enough to benefit from programmed control. Today's home electronics devices support a wide range of options and controls.

At the same time, the emergence of personal digital assistants has created new possibilities for programmed device control. Basic PDA's can dial stored phone numbers. More advanced ones can also send messages to computers or facsimile machines, control remote devices using tone dialing, or even store and play back infrared control sequences such as are used for controlling televisions and other consumer audio/video devices.

Unfortunately most devices are not designed for programmed control. Consumer electronics devices can easily be controlled by a human with an infrared remote control, but only because the human can observe the state of the device and act accordingly. Programmed control units lack state awareness, and accordingly are generally unable to achieve even simple goals.

This paper discusses state awareness problems in building programmable controllers for devices with interfaces designed for interactive use. The next section describes in detail the problems involved with programming a controller for home audio/video equipment. It also formalizes the problems by defining statelock—a condition that inhibits programmability.

The following sections present theoretical results to show that state-awareness problems are fundamental and present design strategies for avoiding statelock. The last section discusses the results and shows how they apply to a different control problem—the use of automatic telephone dialers—and presents other related obstacles to building programmable controls for consumer electronics devices.

THE PROBLEM

"How can this 'universal' remote be programmed to enter 'home theater' mode?" A learning universal remote control unit controls multiple devices (e.g., televisions, video cassette recorders, stereo systems, etc.) by learning the infrared commands from those devices' individual controls. Many units come with sequence programming features to allow the user to define complex operations that are invoked by a single button. A typical goal is to define a "home theater" button that turns the television to a designated channel (typically channel 3 or 4 in the U.S.), turns the VCR on and sets it to display onto the TV, selects VCR input on the stereo system, and sets a moderate volume on the stereo (and no volume on the TV). All together, this
mode would control three devices directly and perhaps others indirectly (selecting VCR input turns off the CD and cassette players on some models of receiver) to provide the type of home entertainment so often advertised!

Unfortunately, there is one problem involved in programming the home theater button: in most cases, it cannot be done. In fact, there are many simpler operations that cannot be programmed. This is not due to the incompetence of the programmer, nor due to the weakness of the universal remote control device. Rather, the problem is one of poor programmability in the devices being controlled.

To illustrate the fundamental problem in programming remote-controllable devices, examine the simplest operation that cannot be programmed on many TV sets. There is no way to program a button to turn on the TV. There is a power button, but it operates as a toggle. It will turn on the TV if it is already off, and it will turn it off if it is already on. This implementation usually works well when a live human is operating the remote control (at worst, an error can be easily corrected), but a program has no way of knowing whether the TV is on or off. For the home theater button, the user would have to either define two buttons, one when starting with the TV off and another when starting with it on, or assume the TV is always on (or off) before entering theater mode (and therefore having to remember to make it so before pressing the magic key).

The problem, simply stated, is that many TVs, VCRs, and other devices have internal states that are not always known to the remote control. Controllable devices are simple finite state machines and remote control units generate command tokens that trigger transitions between states. Figure 1, for example, shows the simplest useful device—one which has two states, on and off. This state machine could support three useful command tokens, ON, OFF, and POWER (a toggle). With ON and OFF, the machine is completely programmable remotely, since commands can force it into a specific state regardless of the state in which the device starts. With POWER only, the machine is not remotely programmable if the starting state is not known. Interestingly, a machine with POWER and either ON or OFF is completely programmable (e.g., with POWER and OFF, OFF turns the machine off, OFF followed by POWER turns it ON) as long as excess intermediate states do not matter.

An actual home entertainment system has many more controls and many more states. Most of these states are made visible to a live user (or can be made visible by using a display function), though many of them are hidden until a transition moves the machine into a more visible state. Figure 2 shows a simplified model of a typical TV with states corresponding to 3 volume levels, three channels, and power and with transitions for volume up and down, channel up and down, and power toggle (adding direct channel access would render the diagram completely unreadable; an alternative representation for complex state machines is presented in [3]). The states shown in grey are “hidden.” They can-
not be distinguished merely by looking at the device. They are not the same state, however, since each encodes a different volume/channel pair.

The problem of remote programmability is the problem of creating a sequence of command tokens (i.e., and input string) that always leave the device(s) being controlled in the same desired state. (several analogous problems are discussed below).

In general, absolute access is easy to program and toggles are hard to program. In an n-dimensional state space, absolute access can be used along each dimension to reach the desired state. Relative access can only be programmed if there are fixed ends (e.g., a minimum and maximum volume) after which the command token causes the device to remain in the same state. Any level can be attained by first attaining a fixed level (e.g., by transmitting a number of VOLUME DOWN tokens larger than the number of volume levels) and then moving to the destination level through relative commands (e.g., VOLUME UP). Relative access in a cycle, including the two-element toggle, cannot generally be programmed.

In practice, device state machines are more complicated than this simple model suggests. Even the basic TV state machine shown in figure 2 has an asymmetry in it. When the power is off, channel and volume command tokens do not cause transitions to other states. Accordingly, any program must first turn the power on and then use volume and channel commands.

More complicated devices have still more complicated state machines. A typical VCR has several toggles with unusual interactions: the POWER toggle, the TV/VCR toggle, the TIMER toggle, and various PLAY/STOP, RECORD/STOP, and PAUSE/SLOW toggles. Figure 3 shows the interaction pattern for the first three of these states in a particular model, leaving out other states and transitions such as channel selection and play/record/stop/pause. Programming the VCR requires a nearly complete knowledge of the state machine, and some programs are still not possible because toggles such as PAUSE and SLOW are time- and context-dependent.

**Problem Summary**

Most devices designed for human control model state machines with internal state and externally defined command tokens. The user selects command tokens based on the externally visible state attributes of the machine. Programmable control, however, is more difficult because the start state of the machine is unknown. Accordingly, many such devices cannot be programmed to reach a specific state.

The term **state lock** is used to refer to this lack of programmability. There are four conditions for state lock:

- Controlled devices must have internal state.
- The remote control program cannot determine the state of the device.
- There is no fixed string of command tokens to bring the device to a known state.
- The device state can change without the program being aware.

The first three conditions were discussed above.

Even with these conditions, however, it is possible to synchronize the device and the remote control program if the program is always made aware of any commands. A programmable remote control for a television, for example, could be synchronized with the television at a certain channel, volume, power status, etc. This possibility leads to the fourth condition which states that external agents can change the device state without the program’s knowledge (e.g., a user can control the device manually or the device state can be altered by environmental conditions such as power failure).

It is also useful to include an assumption that any state can be reached from any other through a sequence of command tokens. Without this assumption, state lock could occur simply by leaving the device in a state from which the goal state is inaccessible. For practical
purposes, this assumption is true for almost any consumer device. Some notable exceptions are discussed at the end of this paper.

The four conditions of statelock are formulated so as to allow programmability to be established by nullifying a single condition. The first condition is fundamental to both the home audio/video example and the telephone example discussed below, and is likely to be true for any interesting device. The other three conditions, however, can be avoided. The next section presents prior theoretical work on determining whether a given device can be forced into a known state with a fixed command string. The following section discusses device and command set design options that avoid statelock by nullifying each of the last three conditions.

THEORETICAL BACKGROUND

While little theoretical research has been done on remote control applications themselves, results in automata theory and its applications can be applied to the remote control problem. The second and third conditions of statelock (i.e., unknown state and no fixed string to lead to a known state) correspond to the distinguishing sequence and synchronizing sequence problems for finite state machines [2].

The distinguishing sequence problem seeks either a string of tokens that generates a different output for each initial state in a finite state machine. Not every finite state machine has a preset distinguishing sequence (and it is a PSPACE-complete problem to find a preset distinguishing sequence). Adaptive distinguishing sequences, which change the input string based on output, can be found in polynomial time and have a bounded length of $O(n^2)$ [6]. Remote control devices cannot generally take advantage of the machines that have distinguishing sequences, even when the sequences are known, since they are not capable of observing and analyzing output.

Synchronizing sequences are strings of input tokens that take a finite state machine to a specific state regardless of the initial state. Any machine with a synchronizing sequence can be programmed by first using the synchronizing sequence to reach the known state and then sending a command string to reach the goal state from the known state.

Not all finite state machines have synchronizing sequences. Further, those synchronizing sequences that exist have a length bounded by $O(n^2)$ which is impractical for use with most current consumer electronic devices (e.g., a mid-range television typically has at least 3200 states: 80 channels * 10 volume levels * 2 power states * 2 mute states). Because of this impracticality, the third condition for statelock can be extended to state that there is no short fixed string of command tokens to bring the device to a known state. “Short” can be defined by context as the number of command tokens that can be transmitted, received, and processed in a suitable time interval (e.g., three to five seconds if a user activates the option, perhaps a minute if the option is timer-activated).

Since finite state machine theory cannot nullify the second and third conditions of statelock, and since it does not address the first and fourth conditions, it is necessary to design more restrictive interfaces or automata to ensure programmability.

DESIGN STRATEGIES

This section presents four design strategies for ensuring programmability of remote controlled devices by avoiding statelock.

Ask and Ye Shall Know

One way to avoid statelock is to allow the remote control program to determine the state of the controlled device. Some high-end video products (specifically frame addressable video disk and video cassette players) provide a two-way communication link (generally serial RS232 communications) between the device and a controlling computer. The command set includes state queries (e.g., what frame is displayed, what is the play/pause/stop status, etc.) that generate replies. Remote control programs operate by querying the device state and sending appropriate commands to reach the goal state.

A full communication interface not only solves the statelock problem but also has other programming benefits. The remote control device can include conditional execution (e.g., eject only if there is a tape or disk loaded) and can be given access to any information available in the device itself.

Full communication interfaces, however, are much more expensive and complicated to implement. They require a two-way communication link between the remote control and the controlled device. Wire links are economical, but they limit portability and mobility. Worse yet, the programming complexity requires that either the remote control or the program itself accurately model the state machine of the controlled device in order to determine the correct sequence of com-
mands needed to reach the goal state.

Riding the Bus

Many consumer-level electronics devices are designed to work together with other devices made by the same manufacturer and to share a single remote control. They communicate through a command bus that broadcasts all significant state-changing actions. One company's consumer audio components, for example, communicate through a wired bus system to ensure that active devices are switched through the receiver and other devices are inactive (i.e., selecting PLAY, either manually or through a remote control, switches the receiver input and also selects STOP on other devices).

Programmable remote control devices could be designed to monitor this command bus. In doing so, they could prevent statelock by preventing the device from changing state without the program being aware.

Unfortunately, there are three major obstacles preventing widespread use of bus-monitoring remote controls: 1) the majority of devices do not yet support command busses, 2) even devices that support a command bus only broadcast commands of interest to other components—volume control and other "local" commands are not broadcast, and 3) bus-monitoring requires the same complexity and wire interface as full communication without presenting any significant long-term advantages.

Taking Control

Centralized control interfaces are a more practical alternative for ensuring that the remote control program is aware of all state changes and thereby avoiding statelock. Centralized control interfaces force all device commands to be routed to the remote control first, and then relayed to the controlled device. Several vendors sell components for assembling these interfaces for home automation. The typical kit includes infrared transmitters and receptors along with computer hardware and software to control the devices. User actions transmit signals to the computer program which then formulates an implementation and transmits commands to the individual devices. Interference between user-computer and computer-device communications is avoided either by physically isolating the infrared receptors of the devices or by using different frequencies and patterns. Users also must be prevented from using any manual controls on the device itself.

Centralized control has several advantages. First, it can be implemented to control any device that supports remote-control access. No communication interface or command bus is needed. The remote control system merely learns the commands that the device understands and uses them to control it. Second, centralized control separates human input from programmed control. Since the system requires a general-purpose computer anyway, it is more practical to develop a high-level programming environment. Third, users can use conventional remote control devices for interactive control simplifying the user interface for new users. Finally, centralized control is a well understood model in home automation and can be easily integrated into a home automation system.

Centralized control also has several disadvantages. First, there are situations in which centralized control loses synchronization with the devices. Even when user inputs are reliably directed to the central controller, certain state changes occur directly at the device level. Power failures, for example, tend to force state changes (even when the device is merely unplugged for a few minutes). Similarly, tape and disk player/recorders have physical state corresponding to the presence of (and perhaps writeability of) a tape or disk.

Second, centralized control is inherently non-portable. Centralized control also requires expensive hardware and extensive software and removes the customary manual command access method with which most users are familiar.

Accordingly, centralized control is best suited for environments where home automation is a design priority. Centralized controllers can use two-way communication when available and also provide interfaces to telephone input, sensors, and other home automation components.

Starting From Scratch

The final solution requires redesigning the command set and state machines used in remote controlled devices. It has already been shown that it is neither possible nor feasible to find a reset sequence for the state machine associated with an arbitrary device. It is possible, however, to design machines with easily-accessible reset sequences and more regular state machines. Doing so nullifies the third condition for statelock.

There are three approaches to this solution. First, one can simply define a ground state and implement a RESET command that always causes a transition to the ground state. On a television set, for example, the
RESET might turn the set to channel 2 with the volume at minimum, the mute inactive, and the power off.

A second approach is to redesign the command set to avoid cycles in relative commands. Toggles would be replaced by two commands (i.e., ON and OFF). Larger cycles could either be replaced with absolute access (e.g., entering a channel number numerically) or by removing the cycle (e.g., not wrapping around between the highest and lowest channel). Finally, commands would be available regardless of state (e.g., channels can be changed when the TV is off or when the VCR is playing).

A final alternative is to provide a separate command set for programming from the normal remote control command set. In this way, manufacturers can maintain compatibility with their existing remote control units and user interface while allowing programmable access through a variant command set. A television manufacturer, for example, could provide only a POWER button on the device remote control but control specific phone numbers (e.g., through bar code) so programmable controls could use those commands. This alternative allows the human factors engineers to separate the best human interface from the programming interface.

**DISCUSSION**

These strategies are specifically designed to avoid statelock and thereby improve programmability to address the “home theater” example. In the long run, audio/video equipment will be designed for easy integration with home automation systems. Accordingly, full communication interfaces should become more popular even at the home consumer level. Since full-communication also provides the greatest flexibility and state knowledge (i.e., it can even detect the presence of tapes or disks), it is the preferred long-term solution to this specific problem. In the meantime, centralized control is still appealing for users willing to install complete home automation systems and manufacturers should take steps to provide a more flexible programming command set for users unwilling to do so.

Different problems have different solutions, however, and this section presents a related, but different problem: the use of portable telephone autodialers. After the discussion of autodialers, other non-statelock obstacles to programming devices designed for interactive control are introduced.

**Another Example: Telephone Autodialers**

The problem with programming home audio/video equipment is not limited to televisions and VCR’s. Telephone autodialers face the same problem when trying to define a dialing sequence to reach a certain number from a wide range of telephones. It seems that dialing a telephone is a simple task, and one that is quite amenable to automation. Each telephone line, however, has different state information that affects how an input patterns (i.e., a sequence of digits or tones) is interpreted.

“How can this PDA (personal digital assistant) be programmed to call me at the office” is the analogous lament for autodialers. The goal task as again defined as designating a button (or command) that dials a specific phone number from anywhere that the PDA happens to be. The dialer should also enable calling line identification (a service that allows the recipient to know where the call is coming from) and disable call waiting (a service that allows a second call to interrupt and time-multiplex with the first). The dialer does not know from which phone it is placing the call. Once again, this task is often impossible when the PDA is moved to different phone within a building, let alone the nation or world. Here is an abbreviated list of the obstacles faced:

- Different access codes. Office phones often require “9” or “8” for outside calls.
- Feature codes vary (though most of the U.S. has standardized on “*70” for cancel call waiting and “*67” for enabling calling line ID).
- Feature codes are state-based. Most phone companies treat “*70” as an error if the line doesn’t have call waiting. Some treat it as a toggle to enable call waiting for one call (for a charge). Similarly, “*67” is often an error in areas without calling line ID and is usually implemented as a toggle.
- Toll vs. local call prefixes. International dialing prefixes. In an effort to avoid dialing errors and unexpected phone bills, many local companies require that long distance calls be dialed without a leading “1.” A dialer would need a rate table, possibly based on the phone line’s calling plan, to determine how to correctly dial the call.
- Toll vs. local call prefixes. In an effort to avoid dialing errors and unexpected phone bills, many local companies require that long distance calls be dialed without a leading “1.” A dialer would need a rate table, possibly based on the phone line’s calling plan, to determine how to correctly dial the call.
- International dialing prefixes. The “+” refers to the local countries international dialing access code which varies from country to country.

International portability is out of the question today. Too many countries have phone systems that cannot support advanced features and that may not even sup-
port direct dialing of international calls. Within the United States (and within the North American nations sharing country code 1) and within any other country there should be easier portability. As with home audio/video remote controls, the fundamental problem is that phone lines carry state that cannot be queried by a programmed dialer and that the available command set does not allow the programmed dialer to set the line state with non-toggle commands.

**Autodialers and statelock.** Phone lines suffer the same statelock problems as audio/video equipment. The phone line has state (i.e., it has features, a local calling area, access codes, etc.). The state cannot be determined by the dialer device (i.e., there is no way for an autodialer to ask a phone line what prefix is needed for an outside line or whether call waiting is enabled). There is no fixed sequence of tones that will bring the line to a known state because of toggles (such as "**67")" and error conditions. And, state changes can occur without the dialer being aware, either when the dialer is connected to a new phone line or, less commonly, when a service order or service change alters the line state. Autodialers work fairly well on known lines because they can be made aware of line state and any state changes. They work poorly in a mobile setting because line state varies from line to line (and even the user often is unaware of the state).

**Solutions.** The full-communication solution addresses the state problems facing autodialers. Many telephone line states can be determined with YES/NO questions (e.g., is call-waiting enabled? Is the number xxx-yyyy local?). Some others might require tone responses that could be simply recorded and played back (e.g., what is the international access code? what is the outside-line access code?).

Bus-monitoring is not a useful solution for telephone autodialers. The major state changes occur when moving the dialer between phone lines, and these changes are not broadcast in any form. Similarly, centralized control offers little help both because there is no easy way for the centralized controller to obtain the relevant information and because centralized controllers are not portable and therefore could not be contained in an autodialer. Calling card speed dialing, a popular approach to automatic dialing, is based somewhat on centralized control. Calling card users still must figure out how to reach the central computer (i.e., with the appropriate access codes and toll-free number) before the "automated" part is available. Also, no provision is made for properly billing local calls that are made through calling card speed dialing.

The remaining alternative is redesigning the command interface. A RESET command is one appealing way to ensure a short synchronizing sequence. For a telephone line, RESET could turn off all optional line features (e.g., call waiting) and pre-dial the needed access codes (though feature codes and toll dialing restrictions would still be problems).

A more comprehensive solution would involve redesigning the command set to support an autodialer standard. Here is a rough design for such a command set:

- A prefix code (e.g., "**00#") is defined to indicate that an autodialer is placing the call.
- Following the prefix code is a set of standard feature codes that are absolute (rather than toggle) and never fail (e.g., cancel call waiting is ignored if call waiting is off or not available).
- Next is a universal-form telephone number. Within the U.S., Canada, and other country-code 1 regions, for example, this is a ten digit number. International numbers would be identified by a prefix as is currently done.
- Phone companies would need to recognize the special prefix codes and would have to support a mode where local/toll calling restrictions are eliminated.
- Private phone systems (PBX's and Centrex) would have to interpret the prefix code to include the needed "outside access" codes along with the feature codes. They should also store the entire number before initiating the call so they can determine whether the call is within the local group (or available by low-cost tie lines).

As with audio/video, there are different long- and short-term solutions. Digital telephony (e.g., ISDN) will provide the full-communication system in the long term. In the short term, a RESET command would be most practical, though a special autodialer command set would provide a more thorough solution. In all likelihood, neither will be adopted since ISDN is already being deployed, albeit slowly in some areas.

**Analysis.** The same statelock problems occur in telephone autodialing as in audio/video control. The major differences are: 1) state changes occur due to mobility rather than manual manipulation, and 2) the command tokens and state machines themselves change. Since the second difference can be viewed as a different start state in a larger state machine, it can be reduced to the problem that no synchronizing sequence exists. The same design approaches work in both cases, though the best designs differ due to the different details of the two problems.
Other Obstacles to Programmable Controls

Statelock is not the only obstacle to programmable control of devices designed for human control. Even when statelock is prevented, errors can occur if communications are unreliable. Error correction is generally not provided for human-controlled interfaces because feedback can be used to determine that the error occurred and the user can recover manually (e.g., a missed volume control command can be repeated, an misdialed number can be redialed after hanging up, etc.). Automated controllers cannot usually receive the feedback and accordingly have fewer recovery options available. A phone autodialer may be able to detect a connection and retry the number if no connection is reached in a specified interval. A television remote control generally can do very little unless the command set is based on absolute commands (which can be repeated). In the long run, full-communication interfaces (e.g., serial command connections or ISDN) will provide a mechanism for feedback and error handling. Until then, error handling will be necessarily limited.

A second obstacle is the existence of dangerous states. A VCR should never pass through a recording state on its way from one state to another since recording will overwrite data. Similarly, tape position should properly be considered as a component of state which makes fast forward and rewind operations dangerous since they are not generally invertible. Eject operations are similarly not invertible. Dangerous states can almost always be avoided in a sensible design, but designs should be validated to ensure that dangerous states are not inadvertently entered.

A third obstacle is the real-time nature of these controls. Telephone lines have time-outs and other time restrictions. Audio/video control programs often need to operate against real time constraints (e.g., to record broadcast programs). Accordingly, long term solutions must support real-time components, perhaps through integration with home automation systems.

RELATED WORK ON PROGRAMMED REMOTE CONTROL

Several automation projects have had to deal with problems in programmed remote control. The Home Bus system placed all controls on a central bus, and provided video switching to control television output. It did not, however, incorporate full communication between ordinary devices or otherwise address the problems of statelock.

The IHS system supported a bus onto which infra-red remote controls were attached. It avoided statelock through centralized control, though no formal mechanism existed to prevent users from operating devices manually.

Ironically, much of the work towards making remote controls more user friendly has the side effect of impeding programmability. A menu-driven remote control, for example may simplify select for users controlling devices interactively (since they need use only a simple joystick), but it also requires either a complicated graphical programming system or another command set to support programmability.

CONCLUSIONS

Programmable remote control of human-controlled devices faces many obstacles. The first, and most critical, obstacle is statelock, the inability to maintain synchronization between the control program and the state machine underlying the device being controlled. Statelock can be avoided either by making the remote control aware of machine state (i.e., through a query system or by monitoring state changes from a fixed starting state) or by ensuring that a short command sequence can bring the device to a known state (i.e., a reset command or a well-constructed command set).

In the long term, more extensive communication will be supported between devices and their controllers. Statelock can always be avoided if this communication is available. In the short term, however, the problem of statelock can best be addressed by designing a better command set. Absolute access to states (or sets of states) is better than relative access (e.g., it is better to enter a channel number than to be forced to use channel up and down buttons) and within relative access cycles should be eliminated (including toggles, which are two state cycles). When a modified command set conflicts with other human factors in interactive use (e.g., ease of learning, ease of use, etc.), it can be implemented as an addition to the command set (even a hidden addition) to preserve the original command set.

Whichever approach they take, developers of devices that can be controlled remotely and through automated systems must address the conflicts between their human interfaces and the ability to program their devices.
REFERENCES


Joseph A. Konstan received an A.B. (1987) in Computer Science from Harvard College and an M.S. (1990) and Ph.D. (1993) in Computer Science from the University of California, Berkeley. He is currently an Assistant Professor in the Computer Science Department of the University of Minnesota.

He conducts research on human-computer interaction, user interface toolkits and frameworks, and multimedia systems. He is a member of the IEEE and ACM.