January 1999

Adaptive resource management via modular feedback control

Ashvin Goel

David Steere

Calton Pu

Jonathan Walpole

Adaptive Resource Management Via Modular Feedback Control

Ashvin Goel, David Steere, Calton Pu, Jonathan Walpole
Department of Computer Science and Engineering
Oregon Graduate Institute, Portland

Abstract

A key feature of tomorrow's operating systems and runtime environments is their ability to adapt. Current state of the art uses an ad-hoc approach to building adaptive software, resulting in systems that can be complex, unpredictable and brittle. We advocate a modular and methodical approach for building adaptive system software based on feedback control. The use of feedback allows a system to automatically adapt to dynamically varying environments and loads, and allows the system designer to utilize the substantial body of knowledge in other engineering disciplines for building adaptive systems. We have developed a toolkit called SWiFT that embodies this approach and helps system designers construct, analyze and visualize the behavior of their system. SWiFT provides a framework for composing simple feedback mechanisms that operate within limited domains, and for dynamically reconfiguring them in response to drastic changes in the environment. The result is a system that is efficient and predictable across a wide range of operating conditions. We describe three SWiFT applications to demonstrate the feasibility of this technology.

1 Introduction

Operating systems need to be more adaptive to perform efficient resource management in the face of applications with dynamically varying resource needs running in chaotic, shared environments. Unfortunately, existing approaches to system design result in systems that are unpredictable and tuned to specific operating conditions. We advocate adaptive system software design based on feedback control theory, which has been used in other engineering disciplines to design controls such as fly-ball governors and cruise controls [5].

Although feedback has previously been used for resource management, for example in multi-level feedback schedulers [4] and TCP congestion control [7], the control mechanisms have two problems. First, their design incorporates implicit assumptions about the *range* of their operating environment. For example, TCP's adaptive congestion-control algorithm performs poorly over wireless links. Replacing this policy with one more suited to wireless use results in better performance [20]. Second, these controllers were written without a systematic approach which resulted in an ad-hoc design. This leads to controls that are tightly integrated with the system, which limits their reusability and complicates their maintenance. For example, it would be useful to reuse an existing control, such as parts of TCP congestion control [7], in a new domain such as CPU scheduling [18]. In addition, the lack of a systematic approach makes it difficult to analyze the characteristic behavior of the controlled system. The scarcity of good controllers in system software bears witness to the wizardry required to build them. Our goal is to move the task of building adaptive resource managers to the realm of engineering.

We propose systematic use of feedback control for predictable and controlled adaptation in operating systems. Our approach produces controllers that are *analyzable*, *modular* and *dynamically reconfigurable*. An adaptive system is more predictable when its controls can be analyzed and the controller's operating ranges are known or can be detected. Modularity not only allows reuse but also piecewise analysis of the controls. Further, modular controls are easier to modify when the runtime environment changes. Reconfiguration allows switching simple feedback controls that are tuned to operate within limited domains in response to drastic changes in the environment. Reconfigurable controls thus enable a system to run efficiently across a wide range of operating conditions.

In this paper, we present SWiFT, a software feedback toolkit that embodies our approach and helps system designers construct, simulate, analyze and visualize the behavior of their system. SWiFT supplies tools to analytically or empirically determine the characteristic behavior of a controlled system using a model of the system and a specification of the control goal. This analysis allows us to determine properties such as stability of the system based on control theory. Modularity in SWiFT results from our

This research was supported in part by DARPA contracts/grants N66001-97-C-8522, N66001-97-C-8523, and F19628-95-C-0193, and by Tektronix, Inc. and Intel Corporation.

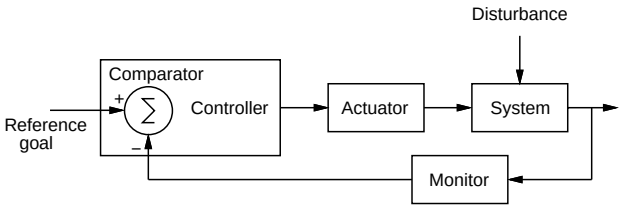

Figure 1: A block diagram of feedback control

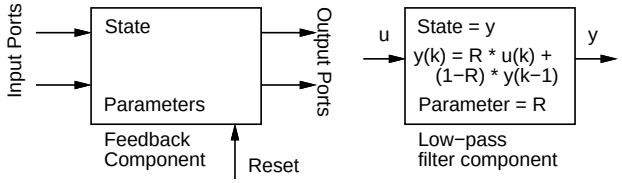

Figure 2: The SWiFT component model, and an example low-pass filter component.

use of components and containers as the underlying abstraction for building controls. SWiFT enables dynamic reconfiguration by limiting the interaction between components to a simple input/output model and by supporting guarding and replugging of controllers [12]. Hence, our approach results in predictable, modular and reconfigurable control designs. In addition, SWiFT provides GUI-based debugging tools such as a software oscilloscope and a library of feedback components such as low pass filters to ease the task of building adaptive system software.

The next section presents the feedback control model in the SWiFT toolkit. Section 3 provides an overview of our approach for designing adaptive applications using SWiFT. We have used the SWiFT toolkit for developing adaptive control mechanisms in a diverse range of domains, from network flow and congestion control in multimedia streams to proportion-based CPU scheduling. This section describes three adaptive applications that were enabled by SWiFT. Section 4 describes the current status of SWiFT. Section 5 summarizes related work in feedback control for designing adaptive systems. Finally, Section 6 presents our conclusions.

2 The SWiFT Model

The SWiFT toolkit follows the design methodology used in hardware control where components interact with each other only through their inputs and outputs, and their behavior can be expressed using block diagrams [6]. Figure 1 shows the abstract architecture of a feedback control system built with SWiFT. The controller adjusts the system to drive system output to match some objective goal. It is integrated with the system through monitors and actuators. A monitor measures the controlled variable, and is the source of the feedback. The controller's output causes the actuator to adapt the system's behavior in response to disturbances, or changes in the system's environment. For example, cruise control monitors the wheel speed and adjusts the throttle when the road incline changes. The SWiFT feedback control design approach separates the system from the control, the monitor, and the actuator, thus providing a modular design.

Components and Containers The basic blocks in SWiFT are feedback components. Feedback components read data from their input port(s), calculate an output value based on their transfer function, and pass the value to their output port. A control circuit is built by connecting a component's output port to input ports of one or more components. Monitors and actuators are special feedback components with no input ports and no output ports respectively. Parameters allow modification of the component's behavior. They are typically adjusted from outside the controller, such as through a slider in the GUI. The state of a component is internal and generally not exposed by the component. A reset port is provided to reinitialize the component's state.

Figure 2 shows the feedback component model and a first-order low-pass filter component as an example. The output of the low-pass filter is an estimator of the average of its recent inputs. The parameter R is an aging factor that determines the contribution of old inputs to the average. The internal state is the previous output of the filter.

Feedback containers, shown in Figure 3, provide modularity and hierarchical structure. A container is a feedback component that contains other feedback components and containers, and defines a circuit of connections among its children and its input and output ports. The container can expose key parameters of its children by mapping its parameter ports to theirs. Figure 3 shows an example of a feedback container that calculates the mean and the standard deviation of an input signal. The parameters of the low-pass filters are exposed by the container.

The outermost or top-level container manipulates and drives the lower layers. Inner components run synchronously to avoid race conditions. A top-level container is either clocked with a fixed rate (the sampling rate) or driven on demand by the system it controls to achieve discrete-event control.

Analysis and Debugging Tools SWiFT currently performs simple analysis for feedback controllers. A component's transfer function is specified by its creator. A container's transfer function is calculated from its layout and

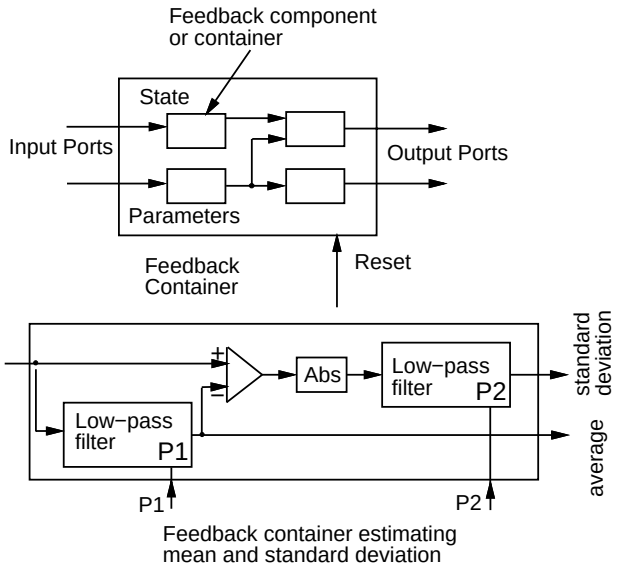

Figure 3: The SWiFT container model, and an example container that estimates the average and the standard deviation. component.

the transfer functions of its children. We use MuPad, a symbolic manipulation package, for doing the algebra.

Along with feedback analysis, SWiFT also helps visualize the outputs of a feedback controller in real time with a GUI toolkit that can be attached to a running control. Other GUI components in SWiFT include a control panel for accessing the parameters of the controller, a scope panel that allows adjustment of the outputs as shown on the scope and various signal generators such as sinusoid, square and random wave generators.

Dynamic Reconfiguration Reconfiguration in SWiFT allows composition of simple feedback mechanisms. It also allows tuning a control circuit's parameters, or replacing some or all of a control circuit at runtime when the operating conditions change significantly. Three types of reconfiguration are possible. First, a component parameter can be altered. Since parameters are constants in a component's characteristic equation, the effect of this component on the controller's behavior must be recalculated. Second, a reset that reinitializes the states and parameters of a component can be issued. For example, the state of a low pass filter that is estimating current latency should be discarded after a network interface switch. Finally, new components can be plugged in or old components can be removed.

Reconfiguration occurs upon the firing of user-specified predicates, called *guards*, that are simple min-max range conditions on the input or the output ports or the state of the controllers. While software reconfiguration in general is complex, it is enabled in SWiFT because of our modular approach. For instance, a SWiFT controller can be replaced by another controller only if both have the same number of input and output ports. In addition, parameters and states of the control can be named and their values may be transferred to the corresponding named entities during reconfiguration.

3 Feedback Control Using SWiFT

As an example of using our approach, consider the task of designing a network flow controller. One starts with a parameterized model of the system's environment, and then designs a control policy that tunes the system's behavior to the model. In this example, the client's received packet rate, C, can be modeled as varying linearly with the server's send rate, S, in the network. If S is less than the network's available bandwidth B, no packets are dropped and C equals S. When S exceeds B, packets are lost due to congestion and C (the client rate) is less than S (and probably less than B as well). The client controller's goal is to tune S to approximate B, by monitoring the rate of packet loss. In other words, the controller will dynamically estimate B and set S accordingly. SWiFT allows monitoring and visualization of the packet loss rate. It also enables modular composition of the controller and its analysis.

Below, we describe controllers that we have built using SWiFT for three diverse system domains: clock synchronization over unreliable networks, a streaming network protocol for multimedia applications and a proportional-share real-rate CPU scheduler.

Clock Synchronization In this application, clients synchronize their clocks with a reference time server. The control goal of this circuit is to ensure that the client time increases monotonically and the client-server phase lag converges to zero. The controller tracks the phase of the reference clock and actuates an adjustable client clock using a phase lock loop (PLL) feedback circuit. The use of a classic controller such as a PLL demonstrates the efficacy of our approach, since PLLs have been widely used in applications such as in a radio receiver for tuning to the carrier frequency of an FM signal.

The phase lock loop works well as long as the network latency does not change drastically over short periods. In the worst case, if the network fails, the client clock can start diverging quickly. We can *guard* the client against this problem. When the client notices that the server has stopped sending signals to it, we can *replug* the control and use a running average of the server time that is *more* accurate than the client clock rate that was set at the last adjustment. Once

the server restarts sending signals, we can replug the PLL back again. Currently, SWiFT provides a framework for implementing guards and replugging. Eventually, SWiFT will use the control goal and controller specification to automatically generate guards to detect unstable behavior and replug a control that is more appropriate for the current environment.

Streaming Control Protocol The streaming control protocol (SCP) [3] is designed for transferring continuous multimedia data over the Internet, while coexisting harmoniously with TCP traffic. Since multimedia applications can tolerate some lost frames but not delayed frames, SCP does not retransmit lost packets. SCP's controller, like TCP, uses exponential back-off and slow start during congestion. However, when the network is not congested, SCP's control goal is to maintain appropriate buffering in the network to achieve predictable latency and maximize throughput. SCP monitors acknowledgement arrival and network buffering and actuates a combined rate and window-based flow control policy that maintains smooth streaming. In contrast, TCP repeatedly increases its congestion window size, causes packet loss and backs off.

SCP's feedback was implemented using SWiFT. SCP replugs its controls when guards fire indicating a change in environment state such as from *steady state* to *congested state*. The SWiFT programming model makes the guards, the controls for each state, and the state transitions explicit and therefore easier to analyze, visualize and modify. Currently, SCP is implemented at the user level. Once SWiFT is ported to the kernel, we will implement SCP in the kernel.

Real-Rate CPU Scheduler The proportional-share real-rate scheduler [18] allocates resources to processes based on an application-specified *progress* rate. Allocating more resources will be wasteful, while allocating less will delay the application. A combination of application-provided hints and application semantics are used to estimate the progress needs of applications. For example, the progress of a producer or consumer of a bounded buffer can be inferred by measuring the fill-level of the bounded buffer. If the buffer is full, the consumer is falling behind and needs more resources while the producer needs to be slowed down.

The feedback controller's goal is to maintain the specified (or estimated) application progress rate. The controller monitors the current progress, say based on the fill levels, and adjusts the resource allocation according to this goal. The control model is challenging not only because different applications have changing progress requirements, but they also have different responsiveness requirements. For example, an isochronous software modem needs a much faster allocation adjustment than a soft real-time media player. In addition, the scheduler must mediate the global allocation requirements of applications at a rate that may conflict with the response needed by each application.

SWiFT helps in building controls with different response behavior, for example, linear or exponential rise or back-off. Moreover, controls can be built with hysteresis so that state change is not too frequent between controls that operate in different environments, such as during CPU underload and during overload. We believe that the various tools provided by SWiFT helped us to build the complex feedback controls in the real-rate scheduler.

4 Current Status

We have implemented SWiFT in C++, C and Java, and we have applied it to several user-level and kernel applications as discussed above. Version 1.0 of SWiFT is available, along with a tutorial, at `http://www.cse.ogi.edu/DSRG/swift`. Currently, we are building a visual editor for designing, implementing, and monitoring controls and dynamic reconfiguration using SWiFT. We are also porting SWiFT to the Linux kernel so that feedback-based kernel allocators, such as a proportional share CPU or disk scheduler [18, 16], can be built entirely within the kernel.

5 Related Work

The ideas in SWiFT are indebted to previous work on feedback-based control systems. Massalin and Pu introduced the idea of feedback-based resource management in operating systems [9] and used it in the Synthesis kernel [14]. Pu proposed a modular approach to building feedback systems [13]. Cen built an early version of SWiFT, and used it to build an adaptive distributed multimedia player [1, 3].

Several commercially available toolkits, such as Matlab [19] support building linear, nonlinear and fuzzy controllers. They provide various predefined control building blocks, simulation, analysis and GUI tools. The target applications of these toolkits are traditional hardware or embedded control systems that have predictable dynamics and gradual transitions. These toolkits are designed to be used off-line at control design time, whereas SWiFT is designed for online runtime use. For example, SWiFT supports direct manipulation of a running control through its debugging tools. In addition, SWiFT supports dynamic reconfiguration through guarding and replugging.

Software feedback has been used extensively for adaptive scheduling, flow and congestion control [7, 17, 8] and intra- and inter-stream synchronization in distributed multimedia systems [15, 2]. The Odyssey system [11] provides efficient

and agile (responsive) resource allocation to mobile applications by using system level resource monitoring and arbitration that insulates applications from insignificant variations in resource levels. These systems implement feedback mechanisms, while we advocate building modular feedback and monitoring policies that can be analyzed, simulated and visualized.

Adaptive systems often use dynamic reconfiguration. For instance, adaptive LFS [10] dynamically chooses the traditional cleaner during low loads and does hole plugging during high loads for overall performance. SWiFT provides support for building such reconfigurable controllers.

6 Conclusions

We have presented SWiFT, a software feedback toolkit that provides a framework for building feedback-based highly adaptive systems using modular composition of simple building blocks. Our experience with the design of controllers for various adaptive applications, including OS resource allocators, indicates that our approach of systematic use of feedback control is valid. It enables building predictable, complex feedback controls that have not been built until now because appropriate feedback analysis and online debugging tools did not exist. Our use of reconfiguration of simple feedback mechanisms allows adaptive systems to operate over a wide environment range while being efficient within each limited domain.

References

[1] Shanwei Cen. *A Software Feedback Toolkit and Its Applications in Adaptive Multimedia Systems*. PhD thesis, Oregon Graduate Institute of Science and Technology, August 1997. Department of Computer Science and Technology.

[2] Shanwei Cen, Calton Pu, Richard Staehli, Cowan Cowan, and Jonathan Walpole. Demonstrating the effect of software feedback on a distributed real-time MPEG video audio player. In *Proceedings of the Third ACM International Multimedia Conference*, November 1995.

[3] Shanwei Cen, Calton Pu, and Jonathan Walpole. Flow and congestion control for internet streaming applications. In *Proceedings of Multimedia Computing and Networking (MMCN)*, 1998.

[4] F. J. Corbato, M. Merwin-Daggett, and R. C. Daley. An experimental time-sharing system. In *Proceedings of the AFIPS Fall Joint Computer Conference*, pages 335-344, 1962.

[5] Gene F. Franklin, J. David Powell, and Abbas Emami-Naeini. *Feedback Control of Dynamic Systems*, chapter 1, pages 1–15. Addison-Wesley, third edition, 1994.

[6] Ibid., chapter 3, pages 111–113.

[7] V. Jacobson. Congestion avoidance and control. *ACM Computer Communication Review; Proceedings of the Sigcomm '88 Symposium in Stanford, CA, August, 1988*, 18, 4:314–329, 1988.

[8] Srinivasan Keshav. A control-theoretic approach to flow control. In *SIGCOMM'91*, pages 3–16, September 1991.

[9] Henry Massalin and Calton Pu. Fine-grain adaptive scheduling using feedback. *Computing Systems*, 3(1):139–173, Winter 1990.

[10] Jeanna Matthews, Drew Roselli, Adam Costello, Randolph Wang, and Thomas Anderson. Improving the performance of log-structured file systems with adaptive methods. In *Symposium on Operating Systems Principles*, October 1997.

[11] Brian Noble, M. Satyanarayanan, Dushyanth Narayanan, James Eric Tilton, Jason Flinn, and Kevin Walker. Agile application-aware adaptation for mobility. In *Symposium on Operating Systems Principles*, October 1997.

[12] Calton Pu, Tito Autrey, Andrew Black, Charles Consel, Crispin Cowan, Jon Inouye, Lakshmi Kethana, Jonathan Walpole, and Ke Zhang. Optimistic incremental specialization: Streamlining a commercial operating system. In *Symposium on Operating Systems Principles*, December 1995.

[13] Calton Pu and Robert M. Fuhrer. Feedback-based scheduling: a toolbox approach. In *Fourth Workshop on Workstation Operating Systems*, pages 124–128, October 1993.

[14] Calton Pu, Henry Massalin, and John Loannidis. The synthesis kernel. *Computing Systems*, 1(1):11–32, Winter 1988.

[15] Srinivas Ramanathan and P. Venkat Rangan. Adaptive feedback techniques for synchronized multimedia retrieval over integrated networks. *IEEE/ACM Transactions on Networking*, 1(2):246–260, April 1993.

[16] Dan Revel, Dylan McNamee, Calton Pu, David Steere, and Jonathan Walpole. Feedback-based dynamic proportion allocation for disk i/o. *Simultaneously submitted to Hot OS*, 1999.

[17] Scott Shenker. A theoretical analysis of feedback flow control. In *Proceedings of SIGCOMM'90*, pages 156–165, September 1990.

[18] David Steere, Ashvin Goel, Joshua Gruenberg, Dylan McNamee, Calton Pu, and Jonathan Walpole. A feedback-driven proportion allocator for real-rate scheduling. In *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation*. USENIX, February 1999. To be published.

[19] The MathWorks, Inc. Matlab product tour. `http://www.mathworks.com/products.html`, 1997.

[20] R. Yavatkar and N. Bhagwat. Improving end-to-end performance of TCP over mobile internetworks. In *Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, U.S., 1994.