The challenges of
delivering a reliable,
easy-to-use
DVR service
to the masses

TiVo-lution

JIM BARTON, TIVO

ne of the greatest challenges of designing a computer system is in making sure the system itself is “invisible” to the user. The system should simply be a conduit to the desired result. There are many examples of such purpose-

built systems, ranging from modern automobiles to mobile phones.

As these products are enhanced over time, however, they often stray from invisibility, imposing new burdens on the user. The two burdens that are probably most annoying to the user are a complex and difficult control interface and lack of reliability. Reliability is not just about whether a computer system functions or not. It encompasses expected behavior, lack of ambiguity, and recovery from unusual events, such as loss of power or network connectivity.

As we began implementation of the TiVo service, we kept these ideas in mind. Our goal was giving viewers a new level of control over their television viewing, while keeping within the context of consumer expectations of a television experience.

In this article I touch on many design facets of the iceberg that is the TiVo service, and the tip of that iceberg most people think of when they hear the name TiVo is the TiVo DVR (digital video recorder).

BACKGROUND

Several trends in electronic and computer technology were under way in late 1997 that we thought would enable creation of a DVR service that could be provided at a reasonable price to consumers:

- Rapidly increasing amounts of hard-disk storage for the same (or lower) price. This was occurring because of brutal competition between hard-disk suppliers to the

PC market. This meant that television viewers could have enormous amounts of video stored locally, creating their own, personal, video-on-demand systems.

- Rapidly increasing sophistication—and declining cost—of integrated chip components. For example, the first inexpensive realtime single-chip MPEG2 encoders were then just coming to market. This allowed our first products to address the largest market at the time, viewers of analog television programming.
- Availability of sufficient open source software components to construct an adequate platform for development of DVR-class software. This meant we could focus our engineering efforts on building the right television experience for the viewer, and thus shorten the development cycle.
- Inexpensive commodity (hardware and software) server technology allowing the deployment of an expansive distributed system architecture.
- Large-scale dial-up Internet access at affordable prices.

Interestingly, there was a great deal of resistance to the concept of a DVR service. Examples include:

- Television networks and other distributors were fearful of the impact the DVR would have on advertising revenues, since it would now be possible to skip through the commercials.
- Major cable television distributors, and industry pundits in general, were dismissive, claiming that video-on-demand would quickly make the DVR obsolete.
- Consumer equipment vendors and industry pundits were skeptical that a subscription service model would find traction with consumers.
- Program guide suppliers and those hoping to turn the program guide into the “front page to the living room” dismissed the idea that the grid-like program guide was suitable as the main interface for controlling television.

TiVo-lution

- Most consumers, long trained to watch appointment television and live with commercials, didn't know they wanted a DVR service until they used it, making consumer education a significant challenge.

WHY A SUBSCRIPTION SERVICE?

A number of considerations led us to implement our technology as a subscription service rather than as a single-sale, stand-alone product. First, the DVR required constant updating with new program guide information to keep up with changing schedules and lineups. At that time, this had to be done via dial-up access, which has a number of inherent costs. The data itself had to be purchased on an ongoing basis and continuously processed into a form suitable for download. These ongoing costs had to be covered in some way, and a subscription fee seemed both simple and straightforward.

Second, despite our best design work, the DVR would be difficult to set up, and the complexity of television distribution meant that program guide accuracy issues could arise over time. We felt this was best addressed through a professional customer service organization that subscribers could contact as needed. Ongoing customer service became another benefit of the subscription fee.

Third, the subscription revenue stream would allow us to fund ongoing software updates and enhancements to the service.

Finally, a subscription model allowed us to subsidize the cost of the DVR hardware itself, bringing the service within the price range of a much broader audience.

SERVICE DESIGN PRINCIPLES

We followed a number of core principles in the initial design of the TiVo service. "Simple," "reliable," "easy to use," and other stock phrases are not specific enough to drive system architectural choices. Instead, we started with a set of more concrete philosophies in designing the overall system. These included the following:

It just works. This means that the product itself should always work as expected. No surprises. No ambiguity. Not "works most of the time." Not "mostly works." It just works. In the consumer experience, if something doesn't work right the first use, it is usually abandoned. So you need to get it right the first time.

Remember, it's television. Everybody knows how television works (even those who claim not to watch it). Television never stops, even when you turn off the TV set. Televisions never crash. You never need to reboot your television. Television always has perfect lip-sync.

The main interface to a television is the remote control. Especially with a DVR, the remote is something you use all the time. Part of the simplicity and usability of the product is a result of how easy it is to use the remote.

AC power is unreliable. There is no power switch on a TiVo receiver. The receiver must be able to survive the loss of power at any time without losing track of its state, so a power switch is superfluous.

Connectivity is unreliable. We can't assume that we are able to connect to the service back-end at any given instant. Thus, the basic functionality of the DVR should work whether connected or not, transparently to the viewer. As a corollary, we could not build in a dependency on network bandwidth available to the DVR. All data transfer, including eventually video, would be handled through download.

We must control the cost per subscriber of the service. Addressing the average consumer means achieving an extremely low cost of entry to the service. The hardware platform must allow the software to approach 100 percent utilization of the hardware for a given set of functions, allowing for the lowest possible hardware cost. The service back-end must allow us to operationally manage utilization and marginal costs in the same way.

Privacy must be fundamental to the design. People are understandably concerned about their viewing habits being exposed to others or used in unexpected ways. On the other hand, anonymous viewing information can be used to measure general behavior and to improve programming. Therefore, the system must be designed to protect the viewer's privacy while allowing for collection of relevant information.

Security must be fundamental to the design. TiVo provides a service to its subscribers through the TiVo receiver. Because TiVo takes no hardware margin, and in fact loses money on the hardware, it needs to ensure that it receives the subscription fee to cover costs. This makes it necessary to guard against theft of service and against the use of alternative service providers. It should also be possible, when necessary, to protect stored programming from arbitrary redistribution.

PLATFORM DESIGN PRINCIPLES

Given these directives, TiVo embarked on the creation of the platform infrastructure that would deliver an embodi-

ment of these ideals to the user. At the same time, we had to make choices that TiVo would live with for a very long time—possibly for the life of TiVo as a company.¹

THE SOURCE CODE

We wanted to avoid dependence on outside software suppliers at all times by having control of each and every line of source code. This would ensure that TiVo would have full control of product quality and development schedules. When the big bug hunt occurred, as it always does, we needed the ability to follow every lead, understand every path, and track every problem down to its source.

As a corollary, we needed to specify every byte of software that is used in the production system. That meant being able to build the software in the way we wanted, whenever we wanted, over and over and over again.

This path led us to use open source software as a foundation for the system. There are trade-offs for this choice.

Open source software is neither the savior of software engineering nor the scourge of capitalism. Instead, it is a wide field of source-code options, ranging in quality from dismal to the finest kind, and ranging in copyright licensing terms from draconian to devil-may-care. Support is seldom dependable, build strategies vary, coding styles may be impenetrable, and changes can be fast and furious.

The main DVR application is implemented separately and kept proprietary to TiVo to maintain its competitive advantages. This means paying careful attention to the borders between open source and proprietary software and carefully evaluating licensing terms before incorporating open source packages.

THE DVR HARDWARE

The software complexity needed to deliver the simplicity we were pursuing dictated the choice of a sophisticated operating-system approach. We were going to write a huge amount of software, and we needed to manage it, debug it, measure it, and so forth. We also wanted to have a great deal of flexibility for future upgrades, so choosing an embedded approach targeted to run in a fixed amount of nonvolatile memory was undesirable.

In fact, since a disk drive was necessary for the product, and was the most expensive component, we wanted to achieve maximum utilization of the drive. This meant we would be able to lower the hardware cost in another way. Instead of burdening the platform with a large amount of programmable or flash memory to hold the software, we stored the software on disk and provided only enough boot ROM to contain the self-tests and a disk boot loader.

This is in contrast to most other DVR implementations, which still operate in an old-style set-top mode, keeping all executable code in flash memory and treating the hard disk as simply an add-on module.

THE SECURITY FOUNDATION

There are two aspects of managing security within the TiVo service. The first is authentication. Any individual DVR should be able to authenticate itself to the service back-end, while the service back-end should be able to authenticate itself to any DVR. Once authenticated, security policies and encryption can be reliably managed. A public-key encryption system is the obvious choice.

We wanted to make it as difficult as possible, within the economics of the DVR platform, to corrupt the security of any particular DVR. Specifically, even if an exploit is found, it should be equally difficult to perform the exploit on the next DVR, and so forth. Practically, this means that the security system should be based in hardware, and modification of the hardware using “expert” tools should be required to perform the exploit. Each TiVo DVR includes a secure microprocessor to which are delegated all public-key-based operations. This secure microprocessor contains a unique public/private key pair for each DVR, so that there are no global secrets for DVR authentication.

The second aspect of system security is establishing the “chain of trust” for software running in the DVR (i.e., ensuring that no software executes on the DVR except software that has been tested and certified by TiVo).

Starting with the TiVo Series2 DVR, this is accomplished by using a secure ROM for the boot code of the system. The boot code includes a copy of a public signing key held by the TiVo service. The boot code first verifies its own integrity and loads the operating system, which must be signed with the private signing key. Once the operating system is authenticated, it is started. The operating system contains signatures for all executable files (and certain other critical files) that it may execute; before starting any application, it checks that each file is unmodified. If so, then the application code can be loaded.

NETWORK CONNECTIVITY

The obvious choice was to depend on standardized protocols within the Internet protocol suite. TCP/IP is the foundation for all communications, PPP for dial-up connection management, and so forth. This also allows us to take advantage of improving technology and the declining cost of IP-based equipment and networks, easing future scaling of the service.

TiVo-lution

SOFTWARE TOOL CHAIN

The tool chain used for building the software is critical, as it determines the productivity of the engineering team and many aspects of code quality and performance. GCC (GNU Compiler Collection) was our choice, running on a RedHat Linux distribution.

OPERATING SYSTEM KERNEL

Here we chose to use Linux. Many readers may confuse the operating system kernel with the overall package of software that makes up an operating system; in this case we are referring only to the memory-resident kernel itself. Linux has many advantages for the TiVo DVR platform: virtual memory and paging support; integrated TCP/IP protocol support; support for many different microprocessor implementations and drivers for most common hardware components, allowing for a great deal of flexibility in hardware implementation; adequate realtime features for a DVR platform; and full source code with an acceptable license (GNU Public License, version 2).

SYSTEM SOFTWARE

We use *system software* to refer to all other software that is not part of the TiVo proprietary application that is used in the system. Each of these components was specifically chosen based on requirements and licensing, and incorporated into the software build and deployment process. TiVo does not rely on any particular GNU/Linux distribution; instead, it handcrafts its own distribution with just those components required for the DVR platform.

APPLICATION DESIGN PRINCIPLES

When TiVo began, broadband penetration in the United States was limited. Yet, to provide a satisfying experience to TiVo subscribers, we needed to provide a large amount of data to each TiVo receiver about what television was available, preferably updated on a daily basis.

The obvious choice was downloading data over a phone line, but there were serious drawbacks to that approach. Most people do not have a phone jack near their televisions, and this was (and still is) a serious barrier to installing a TiVo DVR. From the TiVo service perspective, dial-up connections can also be costly.

Despite the drawbacks of dial-up networking, we chose

this path. We also believed that eventually broadband and home networks would be available to most subscribers, which would simplify installation of the DVR. This led us to create a distributed application in which a central server facility synchronized information about available programming with each DVR, using a short, periodic dial-up connection. The DVR then used that data offline to make local decisions about what to record and when.

As we expected eventually to scale the number of receivers in use into the tens of millions, the server architecture had to be highly scalable. At the same time, it had to be incredibly efficient; since TiVo would pay for the phone calls, we needed to minimize the amount of data transferred and ensure that most calls were successful the first time. Even when broadband access is used, there are still overhead costs for bandwidth and equipment that must be managed.

PROGRAMMING ABSTRACTIONS

Aside from the disk drive, the best way to control hardware cost is to limit the amount of semiconductor memory needed in the system. This includes ROM, flash, and precious DRAM. For performance-critical software, which in TiVo means all software involved with user interaction and video recording or playback, minimizing memory use and processor cycles forces the choice of a low-level optimizing programming language.

We chose to use a subset of GNU C++. We did not use exception handling, which generates a great deal more code—code that *should* be rarely executed in a well-designed application. Exceptions can be thrown for many reasons, and the proper path for recovery from them is not always obvious at different abstraction levels. This makes designing a predictable application difficult.

Instead, all object methods that can fail for some reason must return a status code, and the caller must check this code and properly respond to it. A method returning a failure code must leave the object in a well-defined state, such that the caller can respond to the failure in a predictable way. Our testing framework includes ways to inject failure codes and to check for proper handling of them. In fact, we modified the C++ compiler to verify that callers actually *do something* with the status code.

Our fundamental abstraction layer is called the TMK (TiVo Media Kernel). This body of software provides for typical functionality: memory management, threading, synchronization, interprocess communication, etc. It also provides the fundamental object abstractions from which all other application software is derived; examples include templates for lists, arrays, and hashes.

Memory management is a particular focus of the TMK. Memory is the most important resource in the system and key to performance. Mismanagement of memory is also the most likely source of software failures. To minimize memory usage, using shared memory operated on by multiple threads is desirable whenever possible. This raises many issues, however, with synchronization, cleanup, allocation algorithms, etc.

Unbounded memory allocation is certainly possible in a virtual memory-with-paging environment such as Linux, but must be avoided; paging should be a rare event. TMK manages all memory within bounded allocation areas dedicated to various functions. TMK objects are reference-counted and managed with *smart pointers*, (i.e., template classes that behave as regular pointers, but incorporate reference counting). This relieves much of the memory management burden on the programmer.

These allocation areas are themselves objects and incorporate extensive debugging support and consistency checking, including shutdown checks that ensure no dangling objects are left behind. TMK applications are further required to start up and shut down in a regular and consistent manner. This allows incorporation of extensive automatic consistency checking and ensures that memory leaks are found very early in the development process.

Using the TMK abstraction, we ideally expect to be able to create application code that can run forever without crashes and without memory leaks. In practice, we continue to asymptotically approach this unobtainable ideal.

REMEMBERING THINGS

The soul of a DVR is remembering things: when shows come on, what channels a user receives, how long shows should be kept around, and so on. Remembering has to occur reliably, even when the power can drop at any time and the receiver may be offline for days or weeks.

The number of things it needs to remember is enormous. A subscriber using a satellite receiver can potentially receive 1,000 channels; stretched over 24 hours a day for 14 days, this is a lot of program information. Obviously, this data could always be re-acquired from the service; however, that process would be expensive and time-consuming, and it would be visible to the subscriber.

It is also critical that the subscriber's information—channel lists, recording queues, recordings on disk, and viewing preferences—be properly remembered as well.

The structure of the data we need to remember is naturally object oriented. For example, a particular program might appear on several different channels at various times. Popular series can exist as new episodes on major networks or live for years in syndication on local broadcast channels. It is also important that we have strict control over data versioning. A broadcaster at any time might update a show's program information, such as the showing time or episode. We expect the database always to be functional and to run with constant updating as long as the receiver is in use. The database must reliably discard old information and never "lose" any storage capacity.

Because we knew the different object types and their characteristics and relationships in advance, our implementation approach could be optimized for size and speed. We chose to implement a simple object database with auto-generated C++ wrapper classes for every object. Support for simple transactions allows reliable updates of object collections; retrying failed transaction commits is the responsibility of application code. Garbage collection is used to ensure recovery of database memory. These choices give us complete control over functionality and performance, allowing us to create an optimal database implementation for the TiVo DVR.

MEDIA STORAGE

Most of the capacity of the disk drive is dedicated to storing programming. Here, the constraints are very different from those for the database or application software, which typically deals with small transactions with the disk. The most extreme example is high-definition television signals, in which a single minute of video occupies more than 140 MB of storage. In the TiVo/DirecTV HDTV satellite receiver, the DVR must manage recording two HDTV signals at the same time it plays back a third from the disk. In the worst case, this must continue flawlessly in parallel with all other possible functions that the DVR may be performing (e.g., searching for programming using a program guide, downloading program information from the service, writing log files, and paging).

Media storage is managed through a simple file system design that is focused on efficient allocation and performance. The file system is application-managed, not part of the operating system.

Special adaptations have been made to the operating system to support direct input and output to application buffers, bypassing any normal

The soul of a DVR is
remembering things: when shows come
on, what channels a user receives, how
long shows should be kept around...

TiVo-lution

buffering. The disk driver was modified to incorporate hooks to allow the application to specify how disk operations should be prioritized and to allow the specification of deadlines for media storage operations. This way, the application can ensure that the most important operations happen when necessary, but the driver is allowed to interleave other operations so that forward progress is made on all aspects of the application.

USER INTERFACE

It was very important that the user interface to the TiVo DVR be both simple and powerful. Following our core principles, it also had to be a natural extension of a television viewing experience.

The first expression of this was the use of video loopsets as backgrounds to each navigation screen. A video loopset is simply a segment of video that is designed to be seamlessly repeated, giving the appearance of constant, smooth movement.

These loopsets are carefully rendered to be nonintrusive, yet give the impression of a system that is always active. The loopsets, and the overlaid informational graphics, are designed not to be abrasive or pushy; thus, the viewers could take as much time as they wanted to make choices.

The navigational model used throughout is based on a simple list, navigated with a four-direction control: up, down, left, and right. Up and down move a position indicator through the list, moving right provides additional information about the item selected, while moving left takes the viewer to higher-level or previous lists. Appurtenance icons are drawn on the screen, matching icons on the remote control, indicating what actions are available; we called these the *whispering arrows*.

An innovative feature that has been especially popular is the addition of sound cues triggered by remote-control key presses. This affords another way that the viewers can be assured that their key presses are seen and provides instant feedback if a key press is not allowed or an action has been initiated.

We focused a great deal of attention on the design of the remote control.² When watching television using a DVR, viewers naturally use the remote often, perhaps even holding it the whole time. It is important that

viewers not have to take their eyes off their television screens, so the layout and feel of the keys has to be carefully structured to allow navigating the keys by feel. In fact, it should be possible to use the remote control easily in a darkened living room.

Another important aspect of the user interface is that it should be *dynamic*. For example, if the viewer is perusing the list of recorded programs, and a new recording begins, that should be immediately reflected in the interface. This is naturally achieved by implementing the user interface as a view into the object database. The user interface code is informed by the database when items in the current view change. It can then update the presentation for the viewer.

HOME NETWORKING AND BROADBAND EXTENSIONS

Since 2003, TiVo has been integrating networking-based features into the TiVo service as part of a continual process of upgrading the subscriber experience. The central challenge we face is how to continue providing the best interface for accessing and controlling networked services while providing the “TiVo-like” reliability that the viewer expects.

This is especially difficult in a networked environment because of the complexity of home networks and broadband access. Ideally, the consumer should be able simply to plug the DVR into the home network and access available services through obvious extensions to the user interface. Wireless networking based on the IEEE 802.11 standard is an especially thorny issue; despite the impression given by equipment manufacturers and the news media, such networks are difficult for the consumer to get working properly and have a great deal of connectivity and bandwidth variability in the home environment. We again rely on Internet protocol standards for configuring the DVR to operate properly on the network, supporting Zero Configuration Networking (Zeroconf) standards for IP address configuration and service discovery (<http://www.zeroconf.org>).

To lower the cost of the DVR hardware, we chose not to provide built-in networking interfaces in early generations of the Series2 DVR. Instead, we provide a low-cost USB interface. This is because we believed that the percentage of DVRs attached to networks would be modest and grow slowly over time. Instead of burdening every hardware platform with an expensive interface, those subscribers wishing to connect the DVR to a home network could buy inexpensive USB network adapters that match their network facilities. Broadband penetration has increased sufficiently that all future TiVo DVR platforms

will include some form of built-in network interface.

Opening up to network-based services means opening up to network-based attacks on the DVR or the data flowing on the network. These threats can take several forms:

- Attacks on services the DVR exposes to the network, subverting weak protocols or causing some form of DOS (denial of service). We handle this by carefully selecting which services we will expose and testing each extensively. We also implement a standard Linux firewall, with rules to limit DOS attacks.
- Data-based attacks on the DVR—for example, using carefully selected MP3 streams or JPEG images that might cause a buffer overflow in the decoder, thus crashing the application and causing some form of service denial. Another familiar example is sending corrupted URLs or content metadata. Here, the best avoidance strategy is through rigorous testing and, when leveraging open source developments, choosing stable, widely used, and well-tested packages.
- Network snooping of sensitive data, such as personal viewing information or passwords. We use SSL for access to most network services. Security certificates are distributed through the secure chain of trust from the service back-end.
- DVR-to-DVR program transfers being snooped off the network. The transfers are encrypted, and the encryption keys are exchanged using the public keys of the DVRs involved, requiring use of the embedded security microprocessor to decrypt the keys.

Our ultimate goal is that the interfaces to network services should be as invisible and reliable as other functions of the TiVo service and integrate with the user interface in a clean and obvious way.

As part of implementing network services, we chose to provide to our subscribers a free PC-based application³ that makes the PC appear on the home network as a media server. The subscribers, in addition to services delivered directly over the Internet from TiVo, can publish their own content for playback on the DVR.

Network services implemented to date include: streaming of MP3 audio, display of images in various formats (i.e., digital photos), DVR-to-DVR transfers of recorded programs, Web-based scheduling of recordings, DVR-to-PC and PC-to-DVR transfers of recorded programs, and DVR-to-mobile device transfers using the PC as an agent.

A WORK IN PROGRESS

TiVo has been successful, acquiring more than 4 million subscribers since the company was founded. The company has also been successful in another way: in proving

that both a DVR platform and a subscriber business could be created. As a result, the company faces increasing competition from alternative suppliers, especially the digital television distributors such as cable and satellite operators.

The basic principles used in first developing the TiVo service are still sound. In fact, TiVo is creating a new implementation designed to work within the OCAP (Open Cable Application Platform), targeted at cable television distributors. Comcast has agreed to deploy this new TiVo service as an alternative to its own DVR service solution, an acknowledgment of the perceived quality and value of the technology.

TiVo continues to advance its original service platform with new software functionality. This includes further integration with PCs and mobile devices, new ways of selecting programming, and tighter integration with remote scheduling. The distributed design of the TiVo service has proven to be an effective vehicle for deploying these new services. Q

REFERENCES

1. For a more detailed description of the resulting platform design, see Barton J. 2003. From server room to living room. *ACM Queue* 1(5): 20-32.
2. Hafner, K. 2004. Now preening on the coffee table. *New York Times* (February 19).
3. Limited versions of this application are available from TiVo or third parties for OS X and Linux operating systems.

LOVE IT, HATE IT? LET US KNOW

feedback@acmqueue.com or www.acmqueue.com/forums

JIM BARTON is cofounder, senior vice president, and CTO of TiVo, where he sets the technical vision for the company. His special area of emphasis is on the software and digital video streaming technologies behind the TiVo Service. Prior to TiVo, Barton was president and CEO of Network Age Software, a company that developed software products targeted at managed electronic distribution. He also worked at SGI, where he was the lead system software architect of the Full Service Network in Orlando, Florida. He also served as CTO of Interactive Digital Solutions, a joint venture of Silicon Graphics and AT&T Network Systems created to develop interactive television systems. Barton held technical and management positions at Hewlett-Packard and Bell laboratories after graduating from the University of Colorado at Boulder with a B.S.E.E. and an M.S. in computer science.
© 2006 ACM 1542-7730/06/0400 \$5.00