Downloadable Firmware in Advanced Settop Terminals

Samuel Reichgott

General Instrument, Advanced Network Systems

Abstract

There are many reasons why both cable systems and equipment suppliers benefit from downloadable settop terminals, but there are many decisions to be made in developing a downloadable settop system. Downloadability can take place at several levels of complexity. A downloadable firmware system should be designed for open development. While settops should accept firmware incrementally, this poses a challenge to the system controller and its operator. There are new considerations for reliability, separate from those of classical communications. The design should also cover the use of inband video channels for downloaded firmware, and be mindful of various consumer acceptance issues.

BENEFITS OF A DOWNLOADABLE SETTOP SYSTEM

Both cable system operators and equipment suppliers have strong motivations for deploying downloadable settops. Some of these motivations are identical, or are at least two sides of the same coin. Other reasons driving downloadability reflect the differences in their business priorities. Still, both cable systems and equipment suppliers stand to benefit from the deployment of these new settops.

For the Cable System Operator

Brand Identity Whether at the level of a logo or trademark, menu text, or entire applications, systems can use downloadability to distinguish themselves from their competition. This is especially important when neighboring or overlapping systems use similar equipment.

Revenue Generation Downloadability provides a new opportunities for tiered or individual subscription services. Data for downloaded applications, or entire applications, could be downloaded on a pay-per-use basis.

Low Cost Upgrades When equipment suppliers and third-party developers, or Independent Software Vendors (ISVs), improve or develop new applications, a downloadable settop allows a system to distribute these upgrades to their customers without sending an installer or requiring customers to return settops for upgrade.

For the Equipment Supplier

Speed to Market Previously, a settop's firmware had to be complete, in its entirety, before a single settop could be shipped. Now, a reliable platform and basic settop functions are all that are required to field a downloadable settop. Additional features are added as they are developed, while the basic features are already being sold.

Reduction of Risk Regardless of the number of "focus groups" used to help define a product, there are always customers, marketing, and engineering personnel thinking of improvements. With a downloadable system, equipment suppliers can produce a product embodying their best attempts to satisfy customer requirements, knowing that they can still download the "final version." Needless to say, this is also a valuable tool when it comes to fixing "bugs."

Inventory Consolidation Instead of producing different ROMs for different markets, the equipment supplier can really keep its software soft. No matter what the mix of applications, all similar hardware platforms can have the same ROM and defer the differences to download time.

Flexibility It's this very concept that differentiated the early PCs from word-processors. While both had about the same computing power, the PC won the market, even among single-application users, because of its ability to change. Customers are naturally attracted to the product that promises a hedge against obsolescence.

LEVELS OF DOWNLOADABILITY

Settop downloadability is not an all-or-nothing proposition. Downloadability can take place at several levels, each with its own concerns and complexities. Equipment suppliers need to examine their market requirements and determine which levels of downloadability to include in their product. These levels may be described as follows:

Text Level This is the simplest level of downloadability. No executable firmware changes, and downloaded text (or graphics) replaces default text (or graphics) in existing applications. At this level of downloadability, one can change fonts and logos, customize menus by renaming standard features, display station call letters, or do basic foreign language text substitution. Text Level downloading is a simple, yet powerful, way to provide a customizable user interface.

Data Level As in Text Level downloading, the executable firmware in the settop does not change. Data Level downloading is distinguished from the simpler Text Level by its ability to present a variety of data to the consumer. This level of downloadability has been available for several years in on-screen messaging and downloadable barker applications. More sophisticated uses of Data Level downloading are in today's on-screen program guides, which include database search and sort capabilities, timers for future tuning and (where supported by the cable system), future purchasing and recording.

Interactive Data Level This is similar to the Data Level, except that at the Interactive Data Level the consumer can exercise control over the data that is downloaded. This level of downloading is typified by the various browsing applications that are beginning to appear. It will continue to gain popularity as the internet becomes accessible from more settop terminals. Another current example of Interactive Data downloading is the Infra-Red Blaster application, in users select their VCR manufacturer and model, and the settop loads the appropriate data to control its built-in IR Blaster. Note that two-way interactivity is not strictly required for a system to operate at the Interactive Data Level; as long as users can control which one-way data is loaded, the requirements for this level are met. Without two-way interactivity, the user must select from a limited number of pre-programmed data streams. As true two-way interactivity is added to the system, consumers may select more precisely the data that enters their households.

Application Firmware Level This is the first level at which executable firmware is changed. Application Firmware Level downloading is the most powerful tool available to achieve the cable system operator's goal of brand identification. The entire look-and-feel of the settop can be changed, but that is the extent of the change allowed. Downloaded application firmware satisfied with the settop platform's fixed capabilities.

Platform Firmware Level Settops that are downloadable at the Platform Firmware Level can actually change the services available to application firmware. New device drivers or

other platform services can be loaded to enhance communication capabilities on existing hardware. Depending on how much of the platform is downloadable, part or all of the Application Programmer's Interface (API) can be modified. Downloadability at the Platform Firmware Level is essential to meeting the equipment supplier's speed-to-market goal because it allows a generalized hardware platform to be fielded before the details of its use have been fully realized.

The remainder of this paper discusses the issues associated with Application and Platform Firmware Level downloading. While Text, Data, and Interactive Data Level downloading are powerful tools with their own interesting concerns, these are becoming commonplace in the industry. The ability to download and change executable firmware in a settop requires unique solutions to business and technical questions that warrant their own discussion.

OPEN OR CLOSED SYSTEM?

The decision to offer an open platform for downloading firmware is equally a business and an engineering issue. Entirely new business problems arise, such as with whom to team, how to market ISV applications, how to organize for ISV technical support, and how much proprietary information to share. These issues have long been understood in the personal computer arena, but are novel in the cable settop market. In addition to the development required by cable operators, the computer-savvy ISVs require still more development to match their visions of advanced user interfaces and interactivity. Many would argue that these are good problems to have but they are, nevertheless, serious challenges to a business organization built on the traditional cable industry model. Still, it is difficult to argue with the decision to provide an open platform, given the advantage of hindsight in the example of the PC software industry in the 1980s.

Given the business decision to be open, it is the engineering department's job to respond. Here are a few of the areas that must be addressed when making the leap from proprietary to open development.

The API ISV applications need an interface to the settop platform. The APIs should be provided and documented by the settop manufacturer's firmware development team. Minimally, the API hides the settop's hardware details, making it easier for ISVs to write their applications; but this isn't its only benefit. Even if the API is only a firmware-to-hardware interface, it reduces code size and improves reliability because this interface appears only once, and may be thoroughly tested.

While absolutely necessary, a hardware interface API is probably insufficient. Higher-level features should also be provided by the API, with the same benefits of reuse and reliability. Higher level API functions for cable settops might include password maintenance, favorite channel lists, parental controls, downstream data access and upstream data transmission facilities, purchasing, program guide database access, and a host of other features. The biggest challenge, aside from the implementation of these features, is to create a rich enough set of APIs to satisfy applications that will be written in the future.

The Kernel Old-fashioned proprietary code development was often based on the "super-loop" program architecture. A single thread of control would handle each settop operation, one at a time, then start over again at the top of the loop and repeat forever. This was fine for the closed system with at most a simple menu interface; it was easy to get everything done in time. But the super-loop is clearly inappropriate in an open system, where the different operations are variable and unknown when the platform is created.

What is called for is a small operating system; and a multitasking kernel is a good choice. A multitasking kernel allows different jobs (called tasks) to be dynamically created as required. It orchestrates the concurrent operation of all tasks in the settop. Multitasking lets the processing of secure subscription information (program guide data collection, etc.) take place at the same time as downloaded applications monitor the remote control and run the on-screen display. Each of these tasks can be developed independently, with little or no concern for the details of the other tasks in the settop(provided a well documented API is available).

Many off-the-shelf kernels are commercially available, or a custom kernel may be developed. When considering the buy-or-build option, the firmware manager has to ask several important questions: What kinds of debugging tools need to be developed? How long will it take to develop and test? When these questions are seriously examined, the answer should be obvious to all but the die-hard do-it-yourselfer: Buy, don't build. There are plenty of good kernels across the price spectrum. They come with kernel-aware debugging tools, which are already developed and ready to use. They have customer support departments ready to handle your startup problems. Most importantly, they're written by people in the kernel business, not the cable business. Save your development resources for what you do best!

The Development System The development team which creates the settop platform and "native" applications needs tools to do the job. Some of these tools will be purchased, some will be developed by the firmware team itself. Even the purchased tools will have to be adapted to use the settop as a target system. It stands to reason that ISVs will need the same kinds of tools. If it is not in their culture to begin with, the team will have to learn how to package the work they do, creating their toolbench for outside use. This might seem like extra work at first, but it's actually worth the effort because the resulting documentation becomes a lasting reference, which might otherwise not exist.

MONOLITHIC OR INCREMENTAL DOWNLOADING?

When developing a system for downloading firmware, a fundamental question arises. Should the firmware be loaded in a single, monolithic, chunk or in small, incremental, pieces? The answer to this question affects the rest of the design of both the settop loader firmware and the system controller's downloading software.

Fortunately, it is possible to make the monolithic-versus-incremental decision separately for the settop and the controller. If the settop is designed for the more general, incremental loading, the controller can still download all the firmware monolithically. As explained later, this mixed system can have both speed-to-market and usability advantages.

In the settop, incremental downloading promises the ability to add features without disturbing other executing firmware. New kernel tasks can enter the settop, and be started by the loader, even while the consumer is busy changing channels or browsing the guide. Monolithic downloading, on the other hand, assumes that the entire set of replaceable firmware is loaded at once, leaving the settop temporarily unusable or with limited capabilities during load time.

Once the settop is capable of incremental loading, this ability can be generalized from loading firmware, which is stored in non-volatile memory (typically EEPROM), to loading software, which is held in volatile memory (RAM) for a relatively short time. Transient applications can be loaded in RAM, executed, and then erased. This capability can be used for interactive commercials or

infrequently used diagnostics. It can also be a means of cost reduction. Since code is only held in the settop when it is needed, total memory requirements may be minimized.

FIRMWARE REQUIREMENTS FOR INCREMENTAL DOWNLOADING

When firmware is to be incrementally loaded, it is best to assume it can be loaded in any order. While this is easily said, it represents a significant challenge when designing the firmware loader and the downloadable firmware itself.

Relocatable versus Position-Independent

Loading firmware in any order implies that the location in memory where the firmware will reside is unknown when the code is developed. This is true of both the code space, probably in EEPROM, and the data space in RAM. The goal is then to get the code into the settop and allow it to use the platform services without knowing where it will actually land.

There are two ways to achieve this goal. The code can be "relocatable," or "position-independent." Relocatable code is, essentially, incomplete. It carries with it references to other code and data that it requires to do its job, but these references are called "unresolved" and are maintained in a symbolic form. In traditional firmware development, relocatable code modules are linked together at development time and given absolute memory addresses in order to resolve all unresolved references. In a downloadable system, the linking must occur in the target (the settop) because that's where the other required pieces already exist. The settop's loader must maintain all symbols, and their resolving addresses, in memory in case another module is loaded and requires some existing information or function.

Position-independent code, unlike relocatable code, has no unresolved references. Instead, it accesses its own code and data through references relative to values held in the microprocessor's address registers. For example, the code may find a particular variable "at the address that is 10 bytes away from the address stored in the A5 register." Position-independent code, therefore, need not carry symbolic references to its own code or data because the address register values are established after the code and its data have been located in memory. The resulting size of the downloadable file is significantly smaller than the same file in relocatable format. This is important in conserving both downstream bandwidth and settop memory. However, it does not solve all the problems. There still must be a way for the downloaded module to access the functions of other code already existing in the settop, including the kernel. This is the job of the trap libraries, which are explained later.

The decision to go relocatable or position-independent is a trade-off between flexibility and cost. With relocatable code, all relevant symbols are available to all code on an as-needed basis. There is much less need for foresight, because a developer can count on the availability of any function or data to already be in the settop. However, the cost (in terms of downstream bandwidth and memory) to transport and store all the symbols can be enormous. Conversely, position-independent code is very efficient; only the executable code, in its binary form, needs to be downloaded and stored. Great foresight must be used to provide the trap libraries (or other mechanism) required to access all the functions that are already present. If essential functions aren't given the proper visibility, some efficiency and reliability may be lost in having to download the same function more than once. This is the same kind of foresight required when developing the settop's API. Luckily, if the API requirements

analysis is truly complete, the trap library requirements should be well understood.

Trap Libraries

A "trap" is a mechanism by which a code module ceases executing in its own code space, forces the processor to execute code in another module's code space, and then returns to resume execution at the point of the trap call. Most commercially available kernels provide a trap library to access kernel functions. Settop platform developers need to create their own trap libraries in order to provide access to their API. There are other alternatives, but a trap library is a very code-efficient means of providing an API interface when using position-independent downloading.

MIGRATION TOWARD AN INCREMENTAL DOWNLOAD SYSTEM

If the settop platform supports incremental downloading, the cable system controller can still download monolithic code releases. From the controller's point of view, it transmits a single file to its terminals. From the settop's point of view, this single file actually consists of separate pieces of the settop firmware. This approach significantly simplifies the controller's development while leaving flexibility for the future.

One of the concerns for the controller is to track which firmware is loaded in each subscriber's settop. As an example of why firmware tracking is necessary, consider the fixed capacity of each settop's downloadable memory. It would be an error to try to download more code than the settop could hold and expect all the features to function. Unless the controller keeps track of all separately loaded modules in each settop, such errors cannot be avoided. It's relatively easy to group settops with identical hardware capabilities together and download the same firmware to all of them. It's a much more complex tracking problem if each settop can be sent different firmware. It also stands to reason that a monolithic download environment presents a simpler user interface to the controller's operator.

Still, a need for greater variety among tiered feature sets may call for incremental downloading. Given the limited memory capacity of each terminal, it is impossible to load all features and enable them in tiers. Different subscribers may someday require different firmware mixes, as the number of available applications increases. Eventually, the industry will have to respond by solving the associated database and user interface issues.

PROVIDING A RELIABLE DOWNLOAD ENVIRONMENT

For professionals in the communications industry, it is unnecessary to restate the need for a "clean" distribution plant, appropriate data packetizing, and data error detection/correction. These classical problems are well understood and don't require further treatment. However, there are other reliability issues that must be addressed for a firmware download system to work in the cable environment.

No matter how good your plant, no matter how good your error correction, there will still be data errors. Especially in a one-way cable system (an open-loop system) these errors could have dire consequences for downloaded firmware. The firmware loader in the settop has to be intelligent enough to prevent bad code from getting loaded. The controller must periodically repeat the downstream transmission of the firmware so it is guaranteed to be eventually received, error-free.

With the possibility of data errors, there is the possibility of parts of the download being absent from the settop. There are two ways to handle this problem, and both may be necessary to guarantee that the settop doesn't "crash."

First, the loader may have to know the minimum requirements for system startup. After firmware is incrementally loaded, the loader should determine that a minimum functional set of firmware is present before it starts executing any of the downloaded code. Second, the firmware itself has to be fault-tolerant. This means that if any firmware that is executing cannot find an essential library function, a platform service, or some important data, it must come to a graceful halt. Ideally, a partially loaded system functions to the fullest extent possible. Minimally, it must remain intact and eventually complete the download and proceed with full capabilities.

As two-way systems mature, settop firmware loaders must likewise mature. The loader should take advantage of an upstream channel to request missing pieces of the download. It should also report loader error conditions for diagnostic purposes.

OUT-OF-BAND VERSUS INBAND DOWNLOADING

Inband data transmission for Data Level and Interactive Data Level downloading is already the norm. This promises to set a trend for Application and Platform Firmware Level downloading. In the analog video environment, there is enormous bandwidth available on video channels (compared with the limited bandwidth on a typical out-of-band service channel). This is a great motivation for designing a system for inband firmware downloading. With so many video channels, it is possible to put different applications, or tiers of firmware, on different channels. However, there are system-level and consumer acceptance considerations needed in making this design decision.

A major issue is where to find the data. The settop's loader must be provided with some way to tune the correct channel for the firmware it should be loading. A reasonable approach is to provide just the channel tuning information on the service channel and allow the loader to find the firmware on an inband channel using this information.

Once the developers solve the problem of where to find the firmware, the problem of when to load it arises. Consumers simply will not understand, or tolerate, a settop that suddenly changes the channel just to do a firmware upgrade. There's the classical argument stating: "Just do it at 2 in the morning." Unfortunately, this really doesn't cut it with fans of the Late Late Show. A creative solution is called for -- one which the TV-centric, computer-phobic consumer can handle.

CONSUMER ACCEPTANCE ISSUES

As mentioned before, consumer acceptance must be considered in designing a downloadable settop. The biggest problems stem from the limited memory in a low-cost settop. Unless memory is infinite, you'll eventually have to erase some of it to make room for newer, better firmware. And therein lies the problem: Once you erase firmware, the settop just won't be fully functional for a while.

The settop designers need to determine a minimal set of functions that will consumers happy while their settop firmware is being upgraded. Is channel-surfing enough, or does the program guide need to stay intact? It comes down to a decision of what can be spared until the upgrade is complete. These decisions will determine how much ROM is required to hold the minimum feature set, versus how much EEPROM (or other downloadable memory) is provided for upgrades.

Another related problem that must be solved is consumer notification. Naturally, people accept change better if they know it's coming. Depending on how consumers are notified, this could be costly for a cable operator. Will the

consumer read the note at the bottom of the monthly statement? Probably not. Is a direct mailing the answer? Not for many households that simply trash their junk mail. However, if the settop is smart enough to download firmware, why can't it do the notification? An on-screen message application can be used by cable operators to inform their customers of a pending upgrade. A standard message can be addressed to all affected subscribers. It can provide plenty of advance notice, stating the date and time of the proposed upgrade. In addition, an "emergency" message can pop up on the screen an hour before the upgrade is scheduled to begin. Still, operators will have to expect a few phone calls, no matter how hard they try.

Another area of concern is the retention of user preferences and other settings through a download. Users may be given menu selectable options, for example, a time display or channel number display on the front-panel LEDs. If such preferences are erased with the firmware, even a successful download in the dead of night will result in phone calls the next morning. The problem gets worse if programmed timers for future purchases are lost. This means lost revenue to the cable operator, as well as upset consumers. Settop designers must be mindful to create a separate area of memory to hold these preferences and timers so they are not lost when the firmware is erased. The time spent to properly engineer this issue is well worth the effort, because it avoids problems for consumers and cable operators alike.

CONCLUSIONS

Downloadable settops offer advantages to cable system operators and equipment suppliers alike. Cable systems gain the advantage of improved brand identity, revenue generation, and low-cost upgrades. Equipment suppliers benefit from speed-to-market, reduced risk in deploying new features, inventory consolidation, and the ability to market more flexible products.

Downloadability can be realized at several levels. Text or data can be downloaded without changing the executable code in the settop, or the settop can change its feature set by downloading new code. Downloading can be interactive -- achieving maximum selectability in full-featured, two-way cable systems.

An open system has distinct advantages. Many third-party software developers can add features to the settop, making the equipment more marketable without the expense of additional development resources. The development team must provide facilities to make open development easier, including an API and well-documented development tools. An open system may also call for a multitasking kernel in order to make new code easier to plug in.

Settops can download firmware incrementally while the controller migrates from monolithic to incremental downloading. This will eventually be required as more third-party applications become available.

The vast bandwidth available on video channels tempts settop designers to use them for downloading firmware. However, system and consumer acceptance problems must be solved before inband downloading can be implemented satisfactorily.

Finally, the consumer's impressions of the system must be addressed when designing a downloadable firmware system. Resolving these questions is an important key to providing a successful system.