THE USE OF SCALABLE VECTOR GRAPHICS IN FLEXIBLE, THIN-CLIENT ARCHITECTURES FOR TV NAVIGATION

Michael Adams
Solution Area TV, Ericsson

Abstract

Today's subscribers are demanding more and more from their service providers:

- *Personalization: New behaviors from a new generation of "digital natives", (who expect the service to adapt to them!), powerful search capabilities, and recommendations engines.*
- *Communication: Multitasking, social networking, and sharing the viewing experience through chat and instant messaging*
- *Interactivity: Polls, games, and enhanced programming*

And subscribers want all the above services and features to be delivered as a single, integrated service experience across any device, anywhere, and at anytime!

The Internet has shown how to deliver all kinds of services by means of thin-client approaches using the Representational State Transfer (REST) model. Meanwhile, most deployed cable architectures still rely on a "state-full", thick-client approach.

Can thin-client architectures really satisfy large cable system requirements for performance, scalability, high-availability, emergency-alert system requirements, and compatibility with existing interactive application environments such as EBIF?

This paper will show how a thin-client, browser-based approach can:

- *Support rapid development of new applications without the need for new software download to the set-top*
- *Enable personalization of a service to each subscriber's preferences*
- *Allow full customization of the user-interface, including branding*
- *Allow the use of third-party developers, using Web 2.0 service creation methods*
- *Provide set-top independence and multiplatform portability*
- *Decouple CA/DRM certification from new features and applications development*

INTRODUCTION

Today's subscribers are demanding more and more from their service providers. What they want can be grouped into three main categories:

- Personalization
- Communication
- Interactivity

Personalization

- I want my service to adapt to my needs.
- I need recommendations to sort *my* "wheat" from the "chaff".
- I want to watch anything, on any device, at any time, anywhere.

Personalization provides the subscriber with a better experience, tailored to their needs, and creates "stickiness" for the operator.

Communication

- I want to share my experience with my friends in real-time
- I want to interact with social networking sites
- I want to be notified when I have a phone call and to be given an option to pause my movie if I choose to take the call.

Communication is a basic human need. Many subscribers are already using a laptop while watching TV to turn viewing into a social experience. The TV experience can be extended to allow all subscribers to communicate while they are watching TV to a greater or lesser degree, depending on their preferences.

Interactivity

- I want to request more information about products when they are advertised.
- I want to be able to go directly to the movie after I see the promo.
- I want to be able to play along with my favorite game shows
- I want to be able to vote online about important issues.

Numerous studies have shown that interactivity can help to keep subscribers engaged. Interactive programming includes such things as play-along game shows, audience polls, and the like. Interactive advertising is also an important opportunity.

Interactivity is more natural on devices like a mobile phone or tablet PC than the TV, because these devices can allow more natural user-interfaces through the use of touch screens.

User Interface Requirements

In summary, subscribers are ready and willing to extend their TV viewing experience. There is a new generation of savvier, more educated subscribers (often called "Digital Natives" [1]) that desire new features. Nevertheless, TV is still a living room experience and pure web-style navigation doesn't work.

Personalization, communication, and interactivity require a dynamic, flexible, extensible, high-performance user-interface. Moreover, as the subscriber starts to like, and take advantage of new features, the infrastructure that supports them must be scalable. The system cannot slow to a crawl if everyone presses the "red button" at the same time (for example, during a Super Bowl commercial).

CURRENT USER INTERFACE

Today, most set-top box (STB) "guides" are limited in their ability to support the requirements of personalization, communication, and interactivity because of the way they are developed:

- The user-interface logic is embedded into a monolithic "resident application" that is downloaded to the STB. Because any changes require extensive testing before they can be unleashed on unsuspecting subscribers, each subscriber ends up getting the same guide as every other subscriber.
- The user-interface is developed in a low-level language such as c, c++, or Java. Highly-paid software developers are needed to modify the user-interface or to create new applications. Any changes must be designed, approved, developed, tested, certified, and signed-off; a process that can take 6 months or more. The result is expensive applications that arrive to market late or not at all.

Embedded user-interfaces were the only option when STBs had a small memory footprint and limited CPU power. For

example, in 1998, a typical STB had only 1-2 MB of memory and a 27 MHz CPU [2].

Today, a system on a chip (SOC) can provide extensive capabilities, and these limitations no longer apply. For example, the Broadcom BCM7400 includes a "dual threaded 350-MHz MIPS32 with FPU class CPU" [3]. Typical memory footprints are at least 256 MB. Despite these changes, the thick-client model has persisted.

USER INTERFACE ALTERNATIVES

Over the past 15 years the user-interface model has been revolutionized by the Internet and the World Wide Web, and the thin-client model has become more and more popular. Netbooks are now the fastest growing category of portable computing device, requiring only a browser on the client while the "heavy-lifting" is done by servers in the network.

The advantages of the thin-client model are:

- Extensible; new user-interface logic can be introduced at any time merely by linking in additional web pages.
- Dynamic; changes in the application can be made without a STB firmware reload or reboot.
- Rapid authoring of new applications by graphic designers (not software developers) with a much faster, shorter testing cycle. The development cycle of new features can be reduced from months to days.
- High availability and horizontal scalability is achieved by means IP load balancing.

The limitations of the Internet model are:

- Performance – even a small round-trip latency makes the user-interface unacceptably slow if a synchronous execution model is used.
- Reliance on a guaranteed high-bandwidth communications path between the server and client. This model cannot continue to provide basic navigation when the return path is lost.
- Need for high-performance server "farms" to keep up with client transactions.
-

TOWARDS A THIN-CLIENT SOLUTION

The advantages of a thin-client solution are attractive; however the limitations must be overcome to make this a viable model for the TV.

The goal of this paper is to demonstrate that careful system design can yield a thin-client solution with all its advantages and none of its limitations.

The proposed solution is based on the following:

1. Asynchronous execution model
2. Browser-based STB software environment
3. Scalable vector graphics
4. Sophisticated authoring environments that support rapid development of new applications
5. Personalization through different user-interface styles
6. Multi-level caching for better user-interface performance and lower-latency

Asynchronous Execution Model

One of the main limitations of early client-server implementations was that they were synchronous, waiting for one event to complete before starting another one. The effect is that all of the network latencies are serialized and can add up to a slow user-experience.

This problem has been addressed by the AJAX (Asynchronous Java script And XML) client-server model, which supports asynchronous execution. (See Figure 1)

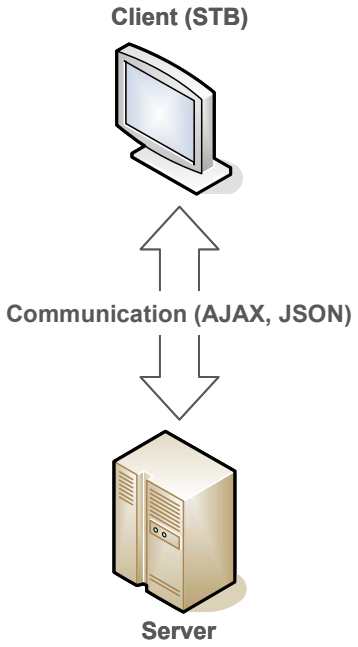

Figure 1: Client-Server Model

The client-side implementation is typically (but not constrained to):

- HTML (HyperText Markup Language) pages, which are navigated by the user as she interacts with the user-interface.
- JavaScript, which provides embedded functions that interact with the Document Object Model (DOM) of the HTML page. Because JavaScript code runs locally in the client environment, user-interface functions can be very responsive.
- CSS (Cascading Style Sheets) allow the format (look and feel) of the user-interface to be separated from the business logic. This allows rapid customization of the graphical aspects (color, fonts, and layout).

Communication protocols are:

- Client-server messages are carried by HTTP using the AJAX format.
- JSON (Java Script Object Notation) is used as data-interchange format.

This standards-based architecture allows each JavaScript object to be transferred asynchronously to the client, eliminating the serialization of transactions and increasing performance of the user-interface.

The server-side implementation is not constrained at all. However, Java Enterprise Edition Server (still commonly known by its old name of J2EE) provides a useful container-based environment.

Browser-based Set-top Box Software Environment

The client that runs on the STB is an advanced web page. Client-server communications are handled by JavaScript and AJAX technology (as previously described). The software framework, both on client- and server-side, is responsible for loading any required JavaScript objects.

The framework also wraps the STB-specific JavaScript Application Programming Interfaces (APIs), enabling new applications to run on any compliant STB.

Advantages of this approach are:

- Allows the use of third-party developers, using Web 2.0 service creation methods.
- Provides set-top independence and multiplatform portability.
- Decouples CA/DRM certification from new features and applications development.

Figure 2 shows the STB software stack. The key points (highlighted on the diagram) are:

- Network Interface – the network interface is based on standard IETF protocols such as SIP, RTSP, and IGMP.

- IMS Applications – IMS (IP Multimedia Subsystem) is used to enable application functions such as presence and messaging. This provides a standards-based way of supporting convergence applications such as caller-id. (IMS is also the foundation of the PacketCable 2.0 specification.)
- Blended Services – the browser runs JavaScript applications and leverages the underlying services layers.
- Characteristics – each STB must meet a baseline set of parameters (such as CPU, memory, and graphics capabilities) to ensure correct performance.
- Browser interface – a set of plug-ins to the browser to allow rapid porting to new STBs.
- Browser capabilities – this will be described in the next section (scalable vector graphics).

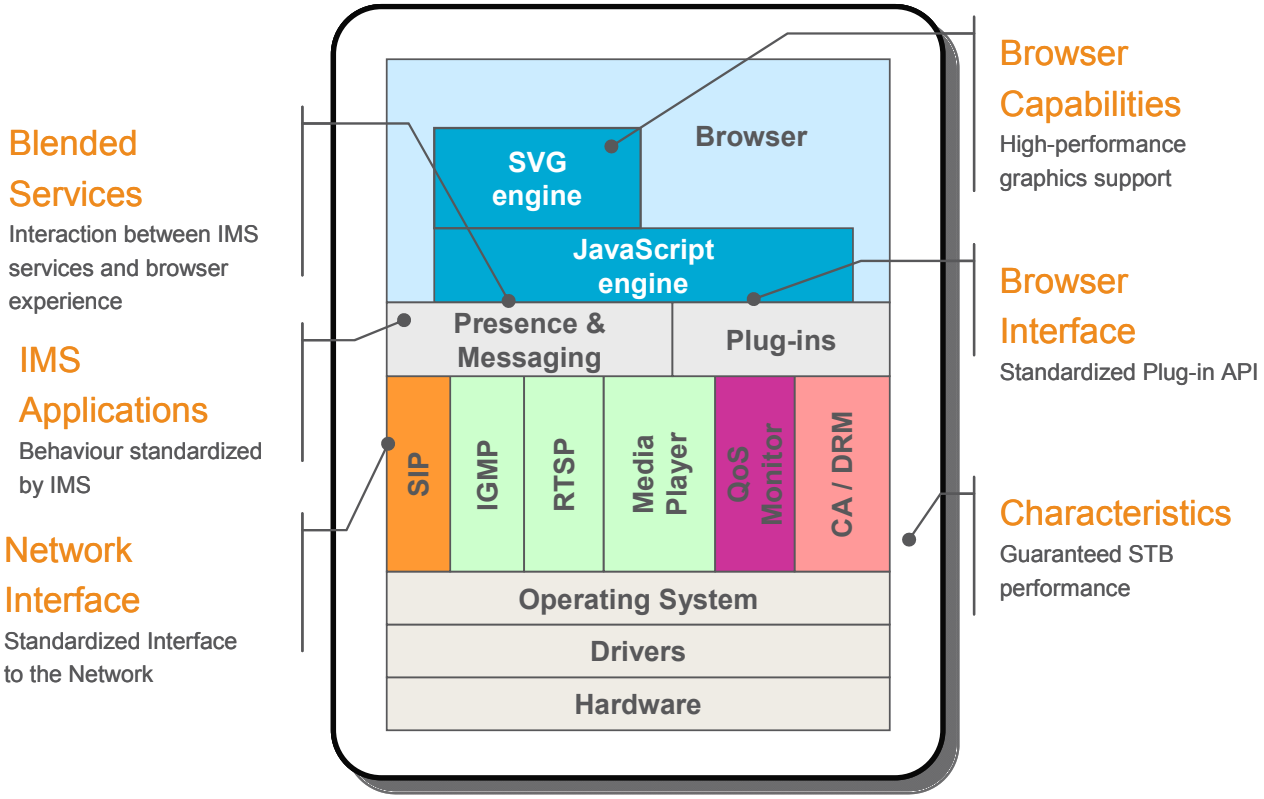

Figure 2: Set-top Box Software Environment

Scalable Vector Graphics (SVG)

SVG is a graphics file format and web development language based on XML. It can be used to create graphically-rich user-interfaces which include animations.

SVG provides similar user-interface functions to Adobe Flash but differs in that it is an open standard that has been under development by the World Wide Web Consortium (W3C) since 1999 [4].

SVG describes two-dimensional graphics and graphical applications in XML. SVG graphics do NOT lose any quality if they are zoomed or resized. Most browsers now support SVG.

Authoring tools

The operator must be able to rapidly introduce new applications, for example (see Figure 3):

1. Casual Games

2. Video Art
3. Real-time Information
4. Web 2.0
5. App Store
6. TV Communities

These example applications can be created by web-applications developers because the environment is identical to that for web development.

Figure 3: Application Examples

Personalization

Different user-interface styles (see examples in Figure 4) can be supported for different user groups or geographical areas.

As previously explained, cascading style sheets (CSS) make it easy to change the appearance and behavior of the HTML pages that define the user-interface.

At the server-side there are other significant advantages to this approach:

1. Different user-groups can have their own unique user-interface style, based on the subscriber profile. For example, a hotel system user-interface may be designed to look completely different from a residential user-interface.
2. Minor changes in appearance and behavior can be made quite simply and rapidly. They are first tested on a lab system, and then with "friendly" subscribers, before being rolled out to the entire subscriber base.
3. Because there is no resident application in the STB, a new user-interface version is published in the same way as updating a web-page. STB firmware downloads and reboots are almost completely eliminated. (The only exception to this is when a browser update is made.)

Figure 4: Personalization of the User Interface

Multi-level Caching

Referring to Figure 5, a multi-level caching scheme is used to improve performance and reduce load on the servers as follows:

1. The browser caches recent HTML pages so that they are retrieved locally if the subscriber goes back to them. The program guide is a good example of this.
2. An HTTP cache stores commonly accessed pages for rapid retrieval without generating any transactions back to the server.
3. A "portal" backend in the server is pre-formatted to increase performance.

It should be noted that all of the cache entries have a time-to-live (TTL) to ensure that information does not become stale. When the TTL timer expires the cache entry is invalidated and the next user request causes a cache-refresh.

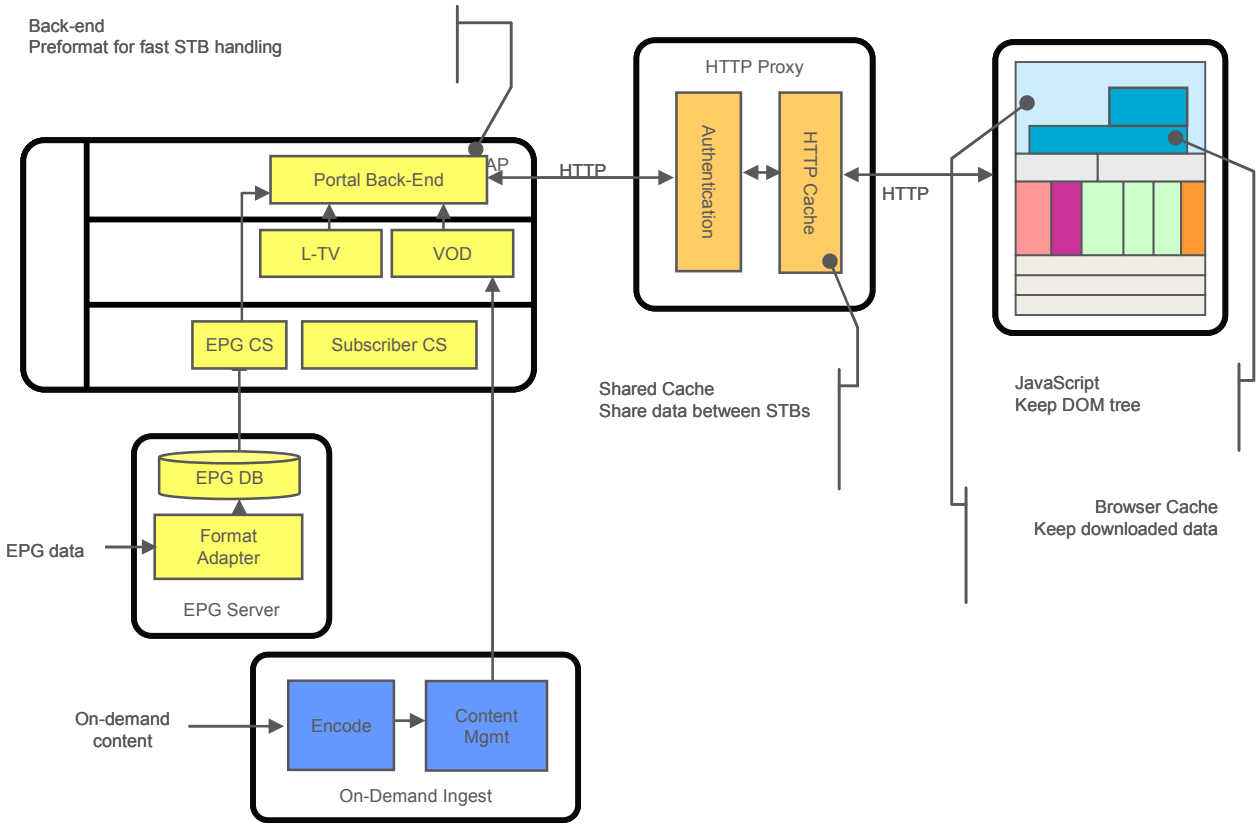

Figure 5: Multi-level Caching Hierarchy

CONCLUSIONS

In this paper we have described how a browser-based, thin-client approach can be used to support a rich, graphical user-interface for the STB.

Making this transition brings with it a number of very significant advantages to the operator:

- Support for rapid development of new applications without the need for new software download to the STB.
- The use of third-party developers, using Web 2.0 service creation methods.
- Personalization of the user-interface.
- Allows full customization of the user-interface, including branding.
- Set-top independence and rapid STB porting through a browser-based approach and plug-ins.
- Decoupling of CA/DRM certification from new features and applications development
- Scalability and performance through the use of multi-level caching strategies.

References

[1] *Digital Natives, Digital Immigrants*, Marc Prensky, 2001

[2] *OpenCable Architecture*, Cisco Press, Michael Adams, page 131

[3] *Broadcom Corporation*, http://www.broadcom.com/products/Cable/Cable-Set-Top-Box-Solutions/BCM7400

[4] *World Wide Web Consortium*, http://www.w3.org/Graphics/SVG/