Extending the XNA Framework for Nomadic Applications

Mark A. Thompson Jr.
San Diego State University, Department of Computer Science
thompsom@rohan.sdsu.edu

Abstract

The profliferation of mobile devices and omnipresent wireless network connectivity has led to an explosion of mobile applications. By exploiting existing technologies and protocols, mobile developers are able to create applications that are both entertaining and useful. Almost in parallel, the availability of video game creation tools has enabled those from outside the video game industry to leverage video game technologies for entertainment and serious applications. This paper describes a prototype application that integrates technologies from two different computing paradigms: Nomadic Computing and video game programming. The primary goal of this application is to expose the challenges of implementing a nomadic computing application.

1. Introduction

Users of mobile devices will soon expect their devices to connect and interact with a variety of applications, beyond those installed on the device. Even though the computing capabilities of mobile devices are increasing, their heterogenity (e.g. different screen sizes, differnt input methods) provide a challenging platform in which to deploy applications. In [1], the author emaphizes how mobile applications have been "dragged kicking and screaming from the desktop, squeezed into ever-smaller devices with tiny screens and diminutive keyboards".
In the context of video games, the above is also relevant. Why should a casual gamer be required to play a game on a small screen with non-standard input devices (e.g. touch screens, small QWERTY keyboards)? There is a movement called pervasive gaming, and in [2] the authors describe games in which players become unchained from the game console and experience a game that is interwoven with the real world and potentially available at any time. However, the hetergeonity of mobile platforms will make this a challenging prospect. This still is an exciting idea, as many mobile devices already provide the benefits of wireless connectivity necessary for such applications.
Rather than describe new protocols and application architectures, Rupert's Virtual World, the game described here, will leverage existing gaming technologies, specifically Microsoft's XNA Framework. The game is a multiplayer game, where "multiplayer" is actually a co-player concept. Only one player is actually playing at a time. The co-players will interact with the current game over a wireless connection from their mobile device. This is a type of nomadic application where the nomads (the co-players) may come and go as they please. The goal of this paper is to describe the initial design challenges of implementating Rupert's Virtual World.
The paper is organized as follows: Section 2 will give an overview of the XNA Framework and nomadic computing, and Section 3 will discuss the techincal challenges and limitations of using those technologies. Section 4 will give an overview of the game itself and it's design. Section 5 discusses related work in video games, and Section 6 will conclude the paper.

2. Enabling Technologies

The following two subsections will give brief overviews of the Microsoft's XNA Game Studio and the nomadic computing paradigm. The majority of material in Section 2.1 is adapted from [3] and [4]. The discussion on the XNA Framework will be kept succint due to the fact most literature on XNA is easily available on theWeb and books from various publishers.

2.1 The XNA Framework

Traditionally, video games have been confined to the domain of expert programmers, advanced computer graphics techniques, high-end digital art, and exorbitant budgets. And as such, video games have for many years have been on their own island of technology and design innovation [13]. However, in 2006 Microsoft introduced the XNA Game Studio, which is a set of tools to make it easier for programmers to write games for Windows and the XBOX 360. It is based on the Visual C# programming language and the .NET Framework 2.0 Common Language Runtime [3]. Underlying the XNA Game Studio is the XNA Framework, which will be described here.
The XNA Framework provides an *Application Model* that contains three essential entities as shown in

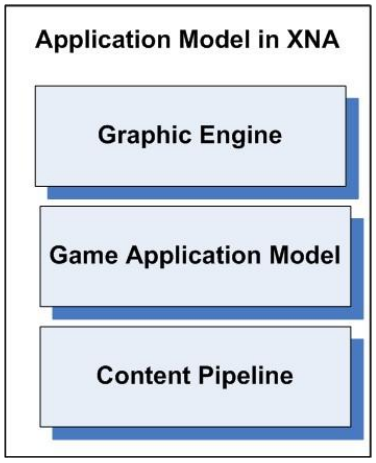

Figure 1. The XNA Framework. Adapted from Riley Publishing

Figure 1. These entities are DLLs (Dynamic Link Libraries), which essentially make calls to the DirectX API [4], and the DirectX API provides direct access to the graphics, audio, and input device hardware. The DirectX interface is as complex as it is powerful, and can be intimidating to novice video game programmers. XNA attempts to hide this complexity so programmers can be more productive.
The Content Pipeline may be the most powerful feature of the XNA Framework. It provides an easy to use way of importing content like textures, 3D objects, music, sound, and other game data from third party sources [3]. In doing so, it greatly reduces the amount of custom code that must be written to do complex graphics and shading [4].
From a programmer's perspective, the XNA Framework uses three important methods, and essentially every game will follow this pattern:

```
// load content, load settings, etc
Initialize()

//called before each frame is drawn, update
// the game time,sound, user input
Update()

//called each frame to draw to screen
Draw()
```

2.2 Nomadic and Ubiquitous Computing

Nomadic computing can be defined as the system support needed to provide a rich set of computing and communication capabilities and services to nomads (mobile users) as they move from place to place in a transparent, integrated and convenient form [6]. The origins of "always connected" and "intelligent" computing systems are often traced to Mark Weiser's seminal article on ubiquitous computing [7]. There, he states that ubiquitous technologies should form the background of the way users expect things to work. Simply said, the most profound technologies are those that "disappear. A currently deployed example is the wallet phone, which allows a cell phone to serve as a digital wallet. In Japan, more than 50 million cellphone users are already using this mobile money

model [Kanipe].
Much of the literature over the years on nomadic computing (also called pervasive computing) has focused typically on consumer and business applications. This may be testament to the fact that academic research in video games is still a new field. Or, it may be due to the fact that consumer and business applications have more viability in terms of commercialization possibilities. For more on the current state of the art of nomadic computing, see the collection of articles from [12].
Before moving on, some clarification is needed on why the implementation of Rupert's Virtual World is worthwhile to pursue. First, the use of a proprietary framework such as XNA provides insight into the issues other nomadic applications face. More specifically, the issues faced when integrating unique (XNA vs. mobile phone) and unrelated (video game vs. wireless connectivity) technologies. The next reason is almost twenty years in the making. In [7], Weiser outlined the technology requirements for a ubiquitous application: low-power computers with convenient displays (modern mobile phones), a network to tie them together, and software systems (the video game). These requirements are common to many of the current mobile device applications, and as such provided the initial inspiration for Rupert's Virtual World.

3. Implementation Challenges

3.1 Network Connectivity in XNA

The XNA Framework does provide wired networking support for multiplayer games, but its use is restricted to the proprietary Xbox LIVE network, or used among other Windows or Xbox machines. Since the XNA Framework is built on the C# language, it is implied that all, or most, of C#'s networking API's are also available inside of an XNA game. This may require a programmer to add a client/server type application into their game (e.g. using sockets), which adds an additional dimension complexity into the game's design and implementation.
As for wireless connectivity over, say Bluetooth or 802.11 (aka Wi-Fi), direct support in XNA is the same as wired connectivity. The worst case would be programming network code using Winsock Win32 C/C++ library, which would result in the reliance on *unmanaged code* in the game.

3.2 Unmanaged Code in C#

A less attractive scenario than those described above is to program network connectivity via the Winsock library available through the Win32 C/C++ API. Winsock adheres strictly to the Berkeley sockets conventions; the de facto standard for network applications [Stall]; most networked application can be programmed by using Berkeley sockets. This same problem can be generalized for all programs written in Win32 C/C++, and since one goal of Rupert's Virtual World is extending XNA, there may be instances where code from outside of C# might be needed. The next question is, how do we get data from a Windows C/C++ program into C#?
Recall from Section 2.1 that XNA is built on something called the Common Language Runtime (CLR). For the purposes of this paper, it can be safely said that any code running inside of XNA (and native to C# lanuage) is considered "managed code" [5]. Managed code, i.e. code running inside the CLR, provides additional application safety through additional runtime error checking and object management. Code that runs outside of the CLR is referred to as "unmanaged code." This includes the Win32 API functions and Winsock for example [5].
Mechanisms in C# do exist to aid developers with dealing with managed/unmanaged code interoperability, however, it is recommended to use the equivalent C#/.NET Framework functionality instead of calling unmanaged APIs [5].

3.3 Game and Network Data Synchronization

Assuming that the issues of connectivity can be overcome, an issue very critical to game play must be addressed: synchronization. In the context of a networked XNA game, this is a crucial issue.
In the XNA context, synchronization, or the lack of, can greatly affect game play. For instance, the XNA Framework by default calls the Update() method 60 times per second. In effect, the screen is cleared and re-drawn 60 times per

second, which provides the illusion of motion. If XNA failed to do this regularly, i.e. became unsynchronized, a game player might experience choppy and erratic game play.
With this in mind, Rupert's Virtual World must ensure that the addition of an additional input device (the network) does not negatively the game experience. For example, if a co-player connects to the game and wishes to interact, by say pressing a button on his/her mobile device, this operation must take care to not freeze game as it waits for network input, and the co-player is given the illusion that his/her input is accepted almost instantaneously. This essentially means care must be taken not affect to XNA's game update mechanisms.
Synchronization is a widely studied topic in the software, distributed computing, operating systems, and networking domains. [9] and [10] are widely recognized in these fields and provide good introductory material on synchronization.

3.4 Development on Mobile Devices

As stated in the Introduction, there is a variety of mobile devices, all with varying physical and technical attributes. This heterogeneity could considered either a limiting, or enabling factor. Limiting in the sense that every device manufacturer provides different programming tools and platforms for development. If the goal is to make Rupert's Virtual World available to a variety of mobile devices, then the requisite programming tools and languages must be learned (time spent), and the programming tools most likely purchased. These sorts of tools are notoriously expensive to the student and hobbyist programmer, just the sort of programmer making games using the XNA Framework. Conversely, if the financial and time constraints are of less concern, Rupert's Virtual World could possibly evolve into a game that many people could be entertained by, simply because there are millions of mobile devices in the world right now.

4. Rupert's Virtual World

4.1 Game Design Approach

The well-recognized game design sequence described by noted game designer Chris Crawford in [11], was employed to some extent in the design of Rupert's Virtual World. Research and preparation was the most important step, since learning the XNA Framework and how it would interact with a wireless device are crucial issues for Rupert's Virtual World. Since XNA frees the programmer the details of graphics, audio, and standard input, this allowed more time to be spent on other issues. More specifically, I/O structure, as this is the most important aspect for Rupert's Virtual World. An interesting historical note: [11] predicts the use of "exotic input devices" in future games. In the context of Rupert's Virtual World, the exotic input devices are mobile phones. So, the inclusion of these exotic devices as I/O mechanisms adds a dimension to the game design process. The over-reaching design goal of Rupert's Virtual World was to design with the limitations of the mobile setting in mind. Rather than work around, or ignore, the limitations and opportunities of mobile devices [1], an attempt was made make Rupert's Virtual World thrive on them.

4.2 Game Concept

Rupert's Virtual World is a non-cooperative multiplayer game. Only one player can "play" at time, but other players, co-players, can interact with game and cause havoc. Our protagonist is Rupert, a relatively normal guy who enjoys walking the globe. Unfortunately, he doesn't know he is our pseudo-superhero, and he does his best when confronted with any situation. For instance, he may be walking the globe, and happen upon a fire, and of course, he decides to put it out. But, the co-players can ruin his day. They are The Antagonistas, a group dedicated to making his life tough. They may decide cause an earthquake, or make televisions with horrible shows playing on them fall from the sky. This multiplayer interaction is the basis of this game. See Figure 2.

4.3 The Co-players

Co-players will be connected wirelessly to a Windows PC running the game. This may either be over Bluetooth or

Wi-Fi. Which to use will be determined by the availability of the programmability of the co-player's mobile device. This will be the most difficult part of the game to implement. In addition, the possibility of managing multiple co-players connected to the game will add another dimension of complexity on top of those discussed in Section 3. However, successful implementation will be the most rewarding aspect of the game since it will truly be an extension to the XNA Framework, and more importantly, this adds the nomadic computing aspect to the game.

4.4 Character Animation

There is no shortage or royalty-free 2D and 3D animations available on the Web and XNA programming texts, which often provide free source code and 3D models. At this stage of implementation, finding the "perfect" character to portray Rupert is not of critical concern.

4.5 The Game Terrain

For the game and level terrain, Microsoft's Virtual Earth web service will be used to provide satellite imagery as game terrain. This is another unique aspect of Rupert's Virtual World. Fortunately, Microsoft's Virtual Earth provides a C# web service API, which allows easy integration into an XNA game.

5. Related Work

The concept that video games can provide more than just entertainment and wasted hours has been embraced by two emerging fields. Firstly, research in the use of video games as educational tools is exploding. The 2006 National Summit on Educational Games is testament to this. This event attracted over 100 experts from fields as diverse as the video game industry, researchers, teachers, and military. This event was put on by the Federation of American Scientists (FAS), and to put the importance of the event into perspective, the FAS has more than 70 Nobel Laureates on it's board of sponsors. The findings of this event can be found in []. Second, the Serious Games Initiative, created in 2002, is yet another effort to broaden the range of disciplines that may find video games useful in their respective fields. A serious game can be defined as the general use of games and game technologies for purposes beyond entertainment [13]. A widely cited game in this field is *Immune Attack*, where players can fight in real "biological battles" inside the human body. The game team included immunologists, instructional designers, game developers, and experts in educational technology. More on this breakthrough game can be found here [14].

6. Conclusion

Advanced graphics, 3D modeling, and audio tools are the primary tools for the commercial video game industry; the availability of rapid development tools, such as the XNA Framework, has the lessened the learning curve for video game creation. In doing so, disciplines and individuals from outside of the video game industry can leverage these tools for academic, entertainment, and creative purposes.
The XNA Framework will allow Rupert's Virtual World to be more than just a video game. Since the complexities traditionally associated with video game implementation are hidden away, time and focus can be spent at a higher level of design. This will allow Rupert's Virtual World to come alive with nomadic casual gamers.

7. References

[1] L.E. Holmquist, "Mobile 2.0", *Interactions*, New York: Mar/Apr 2007. Vol. 14, Iss. 2; pg. 46, 2 pgs

[2] S. Benford, C. Magerkurth, P. Ljungstrand, "Bridging the PHYSICAL AND DIGITAL in Pervasive Gaming", *Communications of the ACM*, New York: Mar 2005. Vol. 48, Iss. 3; p. 54

[3] C. Carter, *Microsoft XNA Unleashed: Graphics and Game Programming for Xbox 360 and Windows*, Publisher Indianapolis, Ind. : Sams Pub., 2007

[4] B. Nitschke, *Professional XNA game programming: for XBOX 360 and Windows*, Wrox/Wiley Indianapolis, IN, c2007

[5] “An Overview of Managed/Unmanaged Code”, *Microsoft Developer’s Network*, [Online], Available: http://msdn.microsoft.com/en-us/library/ms973872.aspx#manunman_rcw, [Accessed: April 26, 2009]

[6] L. Kleinrock,“Nomadic Computing”, unpublished, UCLA Computer Science Department.

[7] M. Weiser, “The Computer for the 21st Century”, *Scientific American: Special Issue on Communications, Computers, and Networks*, September 1991

[8] J.Kanipe, "Payment Via Wallet Phone", *Communications of the ACM*, New York, Jan 2009, Vol. 52, Iss. 1; pg. 13

[9] A. Tanenbaum, *Modern Operating Systems*, 3e, Pearson Prentice Hall, Upper Saddle River, NJ, 2008

[10] W. Stallings, *Data and Computer Communications*, 7th ed., Pearson Prentice Hall, Upper Saddle River, NJ, 2004

[11] C. Crawford, *The Art of Computer Game Design*, Osborne/McGraw-Hill, 1984

[12] *The disappearing computer, Special Issue, Communications of the ACM,* New York, March 2005, Volume 48, Issue 3

[13] B. Sawyer, “Serious Games: Broadening Games Impact Beyond Entertainment”, *Computer Graphics Forum*, Oxford, Sep 2007. Vol. 26, Iss. 3; p. xviii

[14] H. Kelly, et al., “How To Build Serious Games”, *Communications of the ACM*. New York: Jul 2007. Vol. 50, Iss. 7; p. 44

[15] “Summit on Educational Games”, *Federation of American Scientists*, [Online], Available: http://www.fas.org/gamesummit/Resources/Summit%20on%20Educational%20Games.pdf, [Accessed: April 25, 2009]