Towards a Proximal Resource-based Architecture to Support Augmented Reality Applications

Cynthia Taylor and Joe Pasquale
Computer Science and Engineering
University of California, San Diego
La Jolla, CA
Email: {cbtaylor,pasquale}@cs.ucsd.edu

***Abstract*—We are developing a new enhanced cloud-based computing architecture, called the *Proximal Workspace architecture* to allow access and interaction between lightweight devices, e.g., video glasses, earphones, wrist displays, body sensors, etc., and applications that represent a new generation of computation-and-data-intensive programs in areas such as augmented reality. While lightweight devices offer an easy way for these applications to collect user data and offer feedback, the applications cannot be run natively and completely on these devices because of high resource demands. Making these applications available via a cloud, while promoting ubiquitous access and providing the necessary resources to execute the applications, induces large delays due to network latency.**

To solve these problems, we are developing a new system architecture based on supporting workspaces which provide nearby computing power to the users devices and thus mediate between them and the clouds computing resources. Specifically, a workspace provides a set of middleware utilities designed to exploit local resources, and provide specific functions such as rendering of graphics, pre-fetching of data, and combining data from different servers. More generally, the workspace is designed to run any subset of activities that cannot be run on a user's device due to computation speed or storage size, and cannot be run on a cloud server due to network latency. Ultimately, the goal is to produce a set of middleware utilities that when run in the workspace with highly-interactive, computation/data intensive applications will result in better user-perceived performance.

We are exploring this system architecture constructively, by adapting applications that benefit from this architecture and discovering how best to suit their needs. We have already adapted VNC (Virtual Network Computing, which allows local interaction with remote computations) to run under a similar architecture, and as a result increased video performance in high network latency conditions by an order of magnitude. We are currently working on adapting Google Earth to run under this system architecture, with the goal of the user being able to intuitively navigate through renderings of Ancient Rome in video glasses, without being hampered by any bulky equipment.

I. Introduction

In this work, our goal is to make a new generation of programs in augmented reality (and other areas) accessible to lightweight devices. Our work is inspired by three parallel trends in modern computing: the ubiquity of lightweight devices such as cellphones and PDAs, the rise of computation-and-data-intensive programs in areas of ubiquitous computing, augmented/virtual reality, machine learning, and graphics, and the rise of cloud computing. While augmented reality (AR) applications offer a unique and exciting way to integrate computer programs with real life, users are often hampered by bulky equipment [1], [2]. Lightweight devices make it easy for a user to move around while using the device, without being hampered by bulky equipment. And the rise of cloud computing is making it easy to access powerful computational resources from anywhere, while at the same time creating challenging new models of how devices access both data and resources [3].

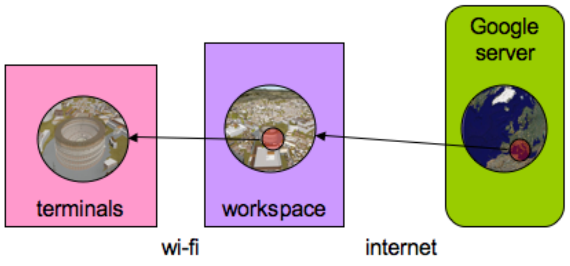

Fig. 1. The workspace stores data that is too large to hold on the lightweight device.

We look at AR applications as presenting a new model for how data is sent between a client and server, and investigate the challenges presented by this new model. In the traditional client-server model, the client will usually request a specific piece of data from the server (e.g. a webpage, image, or multimedia file), and the server will send the client that specific item. In the new model presented by AR applications, the client will send the server a location that the user is interested in, and the server will send back a large amount of information about the area surrounding that location. The user can then interactively explore this data once it is stored on the client, and the client will periodically send the user's new location to the server.

Previously, the idea of sending surrounding data has been used as a optimization for slow data transfers, i.e., pre-fetching surrounding rows in a database. In the AR model, we see two things which make this new, and present new challenges. The first is the nature of the data: a typical AR application may send vast amounts of very detailed multimedia data to the client. The second is the nature of the user's exploration: users physically explore a three-dimensional environment in a non-linear fashion. These two factors mean that the client has to deal with the data it is receiving in a very different way than it did in the past. The amount of data transferred requires significant storage capacity on the device. The combination of non-linear data retrieval by the user and the nature of the applications means that the client device must also do a significant amount of on-demand rendering of the three-dimensional map data, putting high demands on both its video card and CPU.

Moving applications to a server within the cloud in a thin-client fashion induces large delays due to network latency. To solve this problem, we propose a new system architecture whose key feature is the addition of a "workspace" as a low-latency (relative to the client) intermediary between a client and server(s). Figure 1 illustrates how the workspace fits in to this new data model.

The rest of this paper is organized as follows. In Section II, we review related work. In Section III, we present our workspace-based architecture. In Section IV, we present some experimental results, and finally, in Section V, we present conclusions.

II. Related Work

Many AR games incorporate handheld devices, including *AR-Soccer* and *AR-Tennis* [4], [5]. The *Epidemic Menace* AR game, uses both mobile AR units (consisting of a head-mounted display and a notebook computer worn on the user's back), and mobile phones, but the mobile phones have limited functionality compared to the AR unit [6], [7]. The rendering is much more detailed on the AR units, and they are designed to allow players to play by themselves, while the mobile phones require players to play as a team with a stationary player using more powerful equipment. The *MORGAN* AR system offers a separate version, *MORGAN Light*, specifically for use with handheld devices [8]. This version uses less detailed rendering. The *Studierstube* AR framework also features a stripped down version especially for mobile devices [9], [10].

The *Mobile AR4ALL* system and the *BatPortal* system both use thin clients with mobile devices for AR, but they continue to have limited rendering on the mobile devices, as well as less frequent updates [11], [12]. The *AR-PDA Project* combines a mobile device with a server that a does image recognition and returns a processed video stream [13]. None of these systems focus on the issue of network latency.

III. The Proximal Workspace Architecture

A Proximal Workspace Architecture is comprised of three parts, illustrated in Figure 2: (1) *terminals*, the sensors and devices that the user actually interacts with and that provide data about the users location; (2) the *world*, consisting of the users home and/or work computers, web servers, game servers, and anything else the user interacts with through the internet; and (3) the *workspace*, a temporary computing session running on computational resources very close to the terminals, capable of extending functionality of both terminals and the world. We now describe these classes of components in more detail.

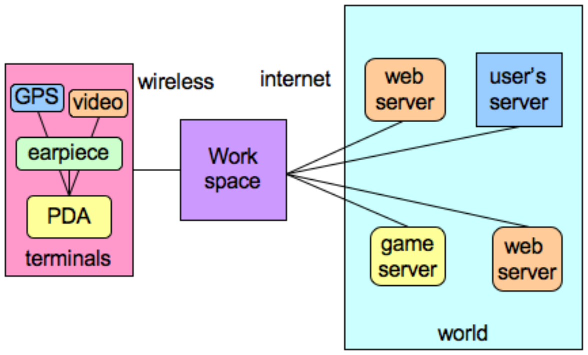

Fig. 2. The system introduces a workspace between the client and servers.

Terminals consist of everything worn or carried by the user of an AR application.They are used for input and output. Input can be specific user actions, equivalent to mouse movements and key presses, or it can be information from sensors such as video cameras, accelerometers, GPS units, thermometers, etc. The terminals may include a hub for local communication and coordination. A set of terminals could consist of a PDA with an integrated GPS and camera, with a set of display glasses worn by the user plugged into it. Terminals form a component similar to the client in traditional thin client systems. Terminals are not required to be capable of anything more than capturing input, displaying output and communicating with the rest of the system. Terminals are expected to be in the system for long periods of time and persist over different workspace sessions.

The world is the set of servers that communicate with the rest of the system through the internet. These will be the game and resource servers for the AR application. These can include a reference website the user is visiting for the first time, or a system server with which they are in semi-constant contact. All persistent state is stored within the world. We make no assumptions about the latency between the terminals and any given server within the world. Some parts of the world may be provided by others and contain large numbers of factors we cannot control. How the world interacts with the rest of the system can have a large impact on system performance and user experience. How long a part of the world remains in the system can very from seconds to decades, and parts can remain consist over different workspace sessions.

Within the workspace there exists a set of middleware utilities that are designed to exploit local resources. These utilities are designed for specific functions, such as rendering of graphics, pre-fetching of data, and combining data from different servers. These utilities could include components of existing AR frameworks, making it easy for designers of AR games to create and distribute their applications [8], [14]. More generally, the workspace is designed to run any subset of activities that cannot be run on the client due to computation speed or storage size, and cannot be run on a server due to network latency. We aim for as much reuse between different applications as possible from these utilities. We define a *workspace server* as the machine the workspaces run on, and a *workspace session* as an individual instance of a workspace interacting with a single set of terminals on a specific workspace server.

Given these distinctions, we define the workspace more precisely as a set of resources given to the user by the workspace server, a machine very close to the user in the network designed to provide systems with these resources. The workspace is used to aid in applications that depend on the users locality, and applications which must quickly communicate with the terminals. The workspace provides local power for computing, memory, and storage, creating the illusion that the terminals have much more power than they do in actuality. It understands its physical location and may be optimized for tasks that are frequently performed at its location. A set of terminals interacts with only one workspace at a time, but may include several different workspace sessions over a given time period.

All workspace sessions are temporary, and how long they last is governed by how long the user is in a set physical location. Depending on their location patterns, users may acquire a workspace session from a workspace server only once, or may frequently obtain workspace sessions from the same set of workspace servers. The workspace can be expected to take in information from many different parts of the world, form them into a consistent whole to present to the terminals, and then intelligently forward input from the terminals back to the world. It is also expected to be able to save persistent state back to the world. The workspace is not expected to be able to run arbitrary code, but will offer a set of gadgets that have been developed to handle common tasks.

While different AR applications differ in design and implementation details, they also have many common components, such as rendering, processing sensor input, and object tracking. For this project, we will develop utilities that fit the common needs of these applications, and discover what utilities can support all systems, and what needs to be more specialized. We will also explore the best system model to support different types of applications, i.e., whether all workspaces should support all application types, or if they should specialize in one application. We plan to leverage existing work in supporting AR applications, including existing AR frameworks.

The workspace has two characteristics that allow it to improve application performance. It is very close to the client in the network, which lets it communicate quickly. It stays in the same geographic location, which it allows it to cache information about its surrounding area. Utilities running on the workspace exploit these two characteristics for better user perceived performance. For example, rendering of games elements or geographic models can be done in the workspace and screen updates can be forwarded to the terminals using VNC, taking advantage of the fast network communication between the workspace server and the terminals. The workspace can cache information that relates to its physical location, e.g. map data, location dependent game elements, etc, and either forward it to the client or access it with internal utilities, creating performance improvements based on its constant geographic location. This option is especially compelling when used for AR applications, which frequently rely on location-related information.

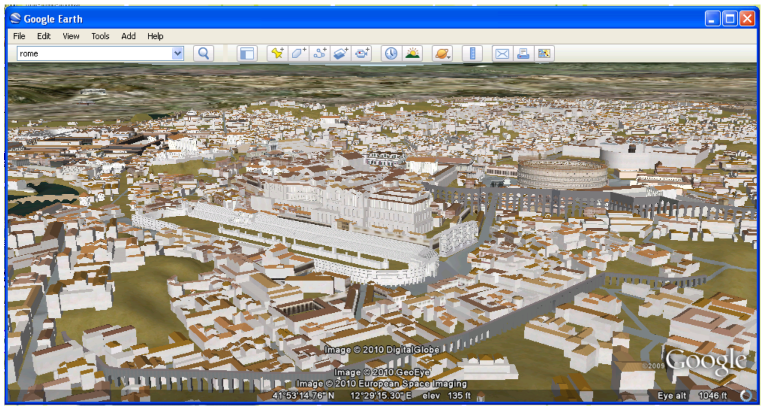
Google Earth
File Edit View Tools Add Help
rome
Image © 2010 DigitalGlobe
Image © 2010 GeoEye
Image © 2010 European Space Imaging
41°53'14.76" N 12°29'15.30" E elev 135 ft
Eye alt 1046 ft
Google

Fig. 3. Google Earth Ancient Rome 3D.

IV. Experimental Results with Google Earth/Ancient Rome 3D

One of our goals for this project is to have utilities that work in conjunction with unmodified applications, rather than rewriting applications to work within our system architecture. Building things to work with unmodified applications offers many advantages to developers and users, including ease of installation and avoiding parallel code maintenance. We will take various approaches to making our system work with unmodified applications, including virtualization-based approaches such as remote display applications and running applications in virtual machines, and intercepting messages sent between client and server applications. An example of the second approach can be seen in our previous work, Improving VNC Performance, in which we add a Message Accelerator which sits between the VNC client and server [15]. No modifications were required to either than client or server to add the Message Accelerator the client behaves as though the Accelerator were the server, and the server behaves as if it were the client.

We are currently working on adapting Google Earth Ancient Rome 3D (pictured in Figure 3) to run under this system architecture, with the goal of the user being able to intuitively navigate through renderings of Ancient Rome in video glasses, without being hampered by any bulky equipment. Google Earth is a perfect example of an application which fits our new data model. The user interactively explores a rendering of ancient Rome, either with a keyboard and mouse or with a more sophisticated input device, and while they explore the Google Server periodically sends a very large amount of multimedia data describing the area they are exploring to the client, which then renders it on a frame-by-frame basis. This puts two burdens on the client: it must be able to store the vast amounts of data being sent, and it must be able to quickly render a complex scene. Google's suggested minimum specs for computers to be able to display Ancient Rome 3d include

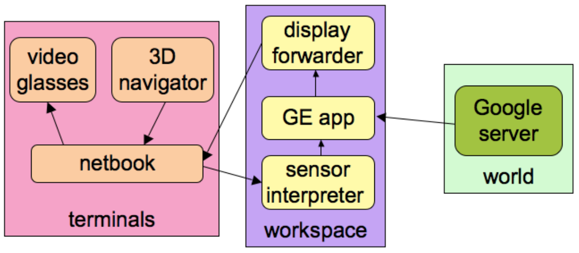

Fig. 4. Google Earth Ancient Rome 3D adapted for the Proximal Workspace Architecture

512 MB RAM, 2 GB of free disk space, Network speed of 768 Kbits/sec or better, and a 3D-capable video card with 32 MB of VRAM or greater. This is beyond the capabilities of a lightweight device such as a netbook or PDA, but by adapting Google Earth Ancient Rome 3D to our system model, we can allow a user to use it on such a device.

In adapting Ancient Rome 3D to our system architecture, we must consider both where parts of the application should be distributed, and what utilities must be created to aid the distribution. The hardware for our test system consists of a Dell Optiplex 755 with a Radeon X1300 video card with 256MB of RAM running Windows XP as the workspace server, with a Lenovo Ideapad S10e Netbook running Ubuntu connected to a 3d Connexion Space Navigator and Video Glasses as the terminals. The Space Navigator is a joystick like device that records both rotation and pressure around the x, y and z axes [16]. Since the Google Earth application cannot meet real time performance demands while running on the netbook, we move it to the workspace. Once Google Earth is running on the work space, we must create utilities to forward input from the netbook to Google Earth, and forward the display updates from the workspace server to the netbook. This is illustrated in Figure 4. In order to forward the display, we use TightVNC [17]. In order to forward input from the Space Navigator, we run a program on the netbook which forwards the raw input from the Space Navigator to the workspace server, where it is then aggregated, translated into units appropriate for Google Earth, and sent to the Google Earth application via API calls.

For our initial results, we measured the frame rate of Google Earth Ancient Rome 3D running natively on the Lenovo ideapad using the Fraps video benchmarking tool [18]. Running natively, Google Earth displayed 0.16 frames per second, resulting in a virtually unusable application. We then ran the adapted Google Earth Ancient Rome 3D on our system, with a link between the server and netbook that had an average round trip time of 0.42 ms, measuring our frame rate by using an instrumented version of Thin VNC. Our version achieved an average frame rate of 7.08 frames per second, resulting in an easily usable application and pleasant user experience.

By adapting Google Earth Ancient Rome to the Proximal Workspace architecture, we allow users to use the application on a lightweight netbook, something that it would be impossible to do running the application natively. Because of the low latency connection between the workspace server and the terminals, the performance of the application is quite good, with no lag between the display of frames. To the user, the effect is as though they were running the application on a much more powerful system, but without being tethered to a desktop machine. We are currently working on adding Motion Node Tilt Sensor to the system, to allow the user to navigate simply by moving their body.

V. Conclusions

We have presented a new architecture to support AR (and other) applications that are highly computationally intensive and that are accessed via lightweight devices, but that are too lightweight to support their actual execution. As these applications are highly interactive, it is imperative that they still execute "near" the user. Consequently, a dynamically allocated workspace that provides computational and memory resources, that is proximal to the user, and that offers a library of utilities that are pertinent to the "AR data model," is the novelty of our design.

In addition to building individual utilities for this architecture, our current work is to also explore how to best design the system as a whole. Moving computation from the client to the workspace adds new issues that must be solved. For example, the workspace must be able to correctly save persistent data at the end of a session with the client, which means it must have a mechanism for figuring out which data should be saved, and which server in the world to save it to. There must be a mechanism for the client to discover and be assigned to a workspace server which is capable of handling its applications. Multiple client sessions within the same workspace machine raises issues of security, privacy, and QoS scheduling. We will leverage existing work when possible when exploring these issues.

Based on our existing work with Google Earth, we feel that the Proximal Workspace system architecture offers unique performance advantages for AR applications and systems, especially those that use the cloud. By adding a workspace, systems can allow users to carry only lightweight equipment, but avoid the costly performance lag of using a pure thin client system with the cloud. In addition, the location-based nature of the workspace system is an ideal match for location-aware AR applications, allowing workspaces to cache information about their own locations and serve it quickly to visiting terminals.

References

[1] W. Broll, I. Lindt, I. Herbst, J. Ohlenburg, A. Braun, and R. Wetzel, "Toward next-gen mobile AR games," *IEEE Computer Graphics and Applications*, pp. 40–48, 2008.

[2] A. Cheok, K. Goh, W. Liu, F. Farbiz, S. Fong, S. Teo, Y. Li, and X. Yang, "Human Pacman: a mobile, wide-area entertainment system based on physical, social, and ubiquitous computing," *Personal and Ubiquitous Computing*, vol. 8, no. 2, pp. 71–81, 2004.

[3] M. Dikaiakos, D. Katsaros, P. Mehra, G. Pallis, and A. Vakali, "Cloud Computing: Distributed Internet Computing for IT and Scientific Research," *IEEE Internet Computing*, vol. 13, no. 5, pp. 10–13, 2009.

[4] V. Paelke, C. Reimann, and D. Stichling, "Foot-based mobile interaction with games," in *Proceedings of the 2004 ACM SIGCHI International Conference on Advances in computer entertainment technology*. ACM, 2004, p. 324.

[5] A. Henrysson, M. Billinghurst, and M. Ollila, “Face to face collaborative AR on mobile phones,” in *Proceedings of the 4th IEEE/ACM International Symposium on Mixed and Augmented Reality*. IEEE Computer Society, 2005, pp. 80–89.

[6] I. Lindt, J. Ohlenburg, U. Pankoke-Babatz, and S. Ghellal, “A report on the crossmedia game epidemic menace,” *Computers in Entertainment (CIE)*, vol. 5, no. 1, p. 8, 2007.

[7] I. Lindt, J. Ohlenburg, U. Pankoke-Babatz, W. Prinz, and S. Ghellal, “Combining multiple gaming interfaces in epidemic menace,” in *CHI’06 extended abstracts on Human factors in computing systems*. ACM, 2006, p. 218.

[8] J. Ohlenburg, W. Broll, and A. Braun, “MORGAN: A Framework for Realizing Interactive Real-Time AR and VR Applications,” *Proceedings of IEEE VR 2008Workshop on Software Engineering and Architecture for Realtime Interacitve Systems*, pp. 27–30, 2008.

[9] D. Schmalstieg, A. Fuhrmann, G. Hesina, Z. Szalavári, L. Encarnaçao, M. Gervautz, and W. Purgathofer, “The studierstube augmented reality project,” *Presence: Teleoperators & Virtual Environments*, vol. 11, no. 1, pp. 33–54, 2002.

[10] D. Wagner, T. Pintaric, F. Ledermann, and D. Schmalstieg, “Towards massively multi-user augmented reality on handheld devices,” in *Third International Conference on Pervasive Computing*. Springer, 2005, pp. 208–219.

[11] C. Geiger, B. Kleinjohann, C. Reimann, and D. Stichling, “Mobile ar4all,” in *Proc. The Second IEEE and ACM International Symposium on Augmented Reality (ISAR01)*, 2001.

[12] J. Newman, D. Ingram, and A. Hopper, “Augmented reality in a wide area sentient environment,” in *Proc. of IEEE and ACM Int. Symp. on Augmented Reality (ISAR 2001)*, 2001, pp. 77–86.

[13] J. Gausemeier, J. Fruend, C. Matysczok, B. Bruederlin, and D. Beier, “Development of a real time image based object recognition method for mobile AR-devices,” in *Proceedings of the 2nd international conference on Computer graphics, Virtual Reality, visualisation and interaction in Africa*. ACM New York, NY, USA, 2003, pp. 133–139.

[14] R. Wetzel, I. Lindt, A. Waern, and S. Johnson, “The magic lens box: simplifying the development of mixed reality games,” in *Proceedings of the 3rd international conference on Digital Interactive Media in Entertainment and Arts*. ACM, 2008, pp. 479–486.

[15] C. Taylor and J. Pasquale, “Improving VNC Performance,” University of California, San Diego, Computer Science and Engineering Department, Computing Science Technical Report CS2009-0943, May 2009, *Submitted for publication.*

[16] “3d Connexion Space Navigator.” [Online]. Available: http://www.3dconnexion.com/

[17] “Tight VNC.” [Online]. Available: http://www.tightvnc.com/

[18] “Fraps.” [Online]. Available: http://www.fraps.com/