Networking Middleware and Online-Deployment Mechanisms for Java-based Games

Chris Carter[1], Dr Abdennour El Rhalibi[1], Prof. Madjid Merabti[1], Dr Marc Price[2]

(1) School of Computing & Mathematical Sciences, Liverpool John Moores University, Byrom Street, Liverpool, L3 3AF, UK
C.J.Carter@2007.ljmu.ac.uk; a.elrhalibi@ljmu.ac.uk; m.merabti@ljmu.ac.uk

(2) BBC Research and Development. Kingswood Warren, Tadworth, Surrey, KT20 6NP, UK
Marc.Price@bbc.co.uk

Abstract. Currently, web-based online gaming applications are predominately utilising Adobe Flash or Java Applets as their core technologies. These games are often casual, two-dimensional games and do not utilise the specialist graphics hardware which has proliferated across modern PCs and Consoles. Multi-user online game play in these titles is often either non-existent or extremely limited. Computer games applications which grace the current generation of consoles and personal computers are designed to utilise the increasingly impressive hardware power at their disposal. However, these are commonly distributed using a physical medium or deployed through custom, proprietary networking mechanisms and rely upon platform-specific networking APIs to facilitate multi-user online game play. In order to unify the concepts of these disparate styles of gaming, this paper presents two interconnected systems which are implemented using Java Web Start and JXTA P2P technologies, providing a platform-independent framework capable of deploying hardware accelerated cross-platform, cross-browser online-enabled Java games, as part of the Homura Project.

Keywords: Web Technologies, Distributed Systems, Java, Homura, NetHomura, Java Monkey Engine, jME, Java Web Start, Deployment, JXTA, Peer to Peer Networking, P2P Games.

1. Introduction

The Homura games project [1] investigated the development of a cross-platform games engine and Integrated Development Environment (IDE) for the development of hardware-accelerated 3D Java games using open-source technologies. This project aims to extend the functionality provided by the Homura Engine, adding systems catering for both the deployment and networking of games built upon the platform, via existing web technologies. Homura is implemented using Java, with the Eclipse IDE and Java Monkey Engine (jME) proving the base for the editor and engine respectively. The deployment platform provides a game portal web application, which facilitates authenticated access to the Homura-based games and securely and robustly deploys the games to the client computers via the Internet. The portal offers features such as automatic updates and the ability to easily enhance existing deployment through additional game content. In addition to the distribution mechanism, the portal also provides an interface which allows users to search for other games, create friendships with other members and instantaneously receive communications regarding the release of new titles on the portal. The system manages game data from in-game experiences such as high-scores and game specific statistics. The networking middleware platform allow developers to easily create Homura-based games which can offer multiplayer, online gameplay in a cost-

effective manner leveraging the resource sharing capabilities of the P2P networking topology, whilst still retaining centralised control over access rights and the maintaining the integrity of persistent game data. This paper describes the motivations for the project, the initial design and analysis phase and the architectural structure of the resulting prototype, named ***NetHomura***. The paper concludes by illustrating a test case of the deployment of a hardware accelerated puzzle game built using the Homura framework in a cross platform manner.

2. Existing Technologies

Online Games portals (such as EA's Pogo [2]) principally use the proprietary Adobe Flash technology to distribute web-based and downloadable game titles through web browsers. These titles commonly offer casual, single player or AI-controlled multi-player gameplay and utilise the omnipresent Flash Player. Whilst the latest version of Flash supports the development of networked online multiplayer games such as Habbo Hotel [3] through its ActionScript language bindings, networking with Flash often requires additional proprietary systems in order to host the system, such as the *ElectroServer* Flash Socket Server.

Another available proprietary technology is the *Unity* game engine.[4] This provides browser based play for Windows and Mac OS X via its custom plug-in for Internet Explorer, Firefox and Safari (implemented as a 3MB ActiveX control for Internet Explorer). In comparison to Flash, Unity provides a much more powerful framework for online games, enabling hardware-accelerated 3D games using DirectX and Open GL to embed within a web page and execute directly within the browser window, or in full screen mode. A similar proprietary technology is being developed by the creators of the Torque Game Engine, Garage Games, called *Instant Action*[5].

Figure 1 illustrates the potential power of the Unity platform and its Tropical Paradise technical demo. Currently however, there is minimal utilization of open-source technologies to provide this level of gaming performance and aesthetic via the Internet.

Figure 1: Tropical Paradise - Unity Demo [4]

3. Key Design Decisions

Prior to implementing the deployment system and networking middleware for the Homura system, the method of deployment was chosen, and a base network API was selected. As the project is to be integrated into the Homura framework, it was crucial to understand the platform and underlying technologies. This section provides a high-level overview of the Homura Engine, and the reasons behind the choices of networking and deployment support.

3.1 Homura and Homura IDE

As stated before, Homura provides both a games engine API and IDE. The engine builds upon the open-source Java Monkey Engine (jME) [6] whilst the open-source Eclipse IDE as base of the games editing facilities. jME was conceived by Mark Powell in 2003 and has developed into a large community based project. Programmed entirely in Java, jME uses the LightWeight Java Games Library (LWJGL) as its low-level OpenGL-based rendering sub-system. It provides a high performance scenegraph based graphics API, which allows for organization of the games entities into a tree structure, where a parent node can contain any number of children nodes, but a child node must contains a single parent. The nodes are organized spatially so that whole branches of the graph can be culled. This allows for complex scenes to be rendered quickly, as typically, most of the scene is not visible at any one time. The scenegraph's leaf nodes consist of the geometry that will be rendered to the display.

3.1.1 jME Features

Whilst jME is an open-source technology, over the last five years it has matured into a feature rich system which is one of the most performant graphical implementation in Java. Key Features that are provided with the Java Monkey Engine are: [7]

- **Bounding System -** All geometry can be enclosed in a bounding system. Boxes, Capsules and Spheres.
- **Collision and Picking -** Efficient bounding volume and triangle accurate picking and collision.
- **Curves -** Bezier curves can be utilised for node control.
- **Effects -** GLSL Shader Support, Vertex and Fragment Program Support (ARB Shaders), Extensible Particle System, Lens Flare, Cloth Simulation, Screen tinting.
- **FastMath Library –** Provides an approximation system for linear algebra using look-up tables.
- **Level of Detail -** Discrete Level of Detail using a switching mechanism for fast model switching. Continuous Level of Detail dynamically collapses triangles of a single model.
- **Lighting System -** Handles up to eight lights at a time with utilities for optimal light selection. jME supports directional light, spot light and point light.

- **Model Loading -** COLLADA, 3DS, OBJ, MD2, MD3, Milkshape, ASE and MD5 support. Supports skin and bones and weighted skeletal animation.
- **Multi-Format Imaging-** Support for BMP, uncompressed TGA, JPG, PNG, GIF and DDS image formats.
- **Shadow System –** Support for Z-Pass Shadow Volumes.
- **State System -** Minimizes OpenGL state changes by tracking and managing the status of the OpenGL machine. Allows for the merging of Texture and Light states within the tree.
- **Terrain -** Terrain blocks act as singular geometry. Terrain Pages implements a Quad tree of Terrain blocks.
- **Texture System -** Supports mip-mapping, environmental mapping, multi-texturing.

3.2 Java Deployment

As the games will be integrated into the Homura Engine and thus jME, they will be developed using Java. Currently, there are two methods of deploying Java-based applications via the internet, Applets and Java Web Start. The Java Web Start technology [8] was chosen as the base platform for the deployment system as it offers several key advantages over using Applets, detailed later in this paper. Java Web Start (JWS) is a standard component of the Java Standard Edition Development Kit (Java SE JDK) and is distributed as a part of the Runtime Environment (JRE). The JRE is required by users in order to execute Java Applications and provides a browser plug-in called the Java Plugin. The Java Plugin is freely available for all major operating systems and browser environments, making the technology ubiquitous amongst desktop PC users. JWS utilises the Java Network Launching Protocol (JNLP) [9] to configure exactly how an application is deployed from a server location to the clients' machines. The protocol uses a simple, standardised XML schema to define several key aspects of the deployment process.

- The appearance of the download window presented to the client upon deployment initiation.
- The creation of desktop and system shortcuts.
- The off-line availability of the application.
- References to any external Java Archive (JAR) files required
- Support for Operating System and architecture specific (e.g. x86/x64/PPC) libraries.
- Dynamic Properties passed as arguments to the application.
- The permissions and security access provisioning of the application.
- Specification of the entry point of the application.
- The configuration of resource updates from the initial deployment location.

3.2.1 Web Start Features and Advantages

An Applet is a specialised Java application, which is embedded directly into the HTML of a web page, and transferred and executed directly on the client machine, within the context of the browser. The browser integration of Applets inherently

limits the ability to run full-screen applications, and also restricts the resolutions at which games can operate. With Applets, the Java Virtual Machine is invoked directly as a process within the browser, and multiple applets must share a JVM instance, which carries a performance and resource overhead. Applets enforce memory restraints that can only be customised by the end user, and are heavily reliant on the security model of the browser for their operation.

JWS applications are executed completely independent of the browser, and each instance is executed within a separate Java process with performance equal to that of standard Java applications. The browser independence means that the application can be adjusted to a variety of resolutions and can run in windowed or full-screen mode with no overheads. An application can be configured by the developer or system administrator to consume additional resources by adjusting the JVM's start-up configuration through the JNLP specification. JWS applications can be easily integrated into HTML pages via a link to the JNLP file, which when executed, invokes the cross-platform Java plug-in, subsequently handling the execution of the application. As a result, JWS exhibits excellent cross platform, cross-browser support. In comparison, the inconsistencies found with the implementation of Applets across browsers and Operating Systems makes the creation of a consistent user experience a difficult task.

Applets require that the game applications base class must inherit from the Java Applet API, which has a defined manner in the way it executes and has to interact smoothly with the containing browser, handling a variety of events, many of which are completely irrelevant to the underlying application. Because applications deployed using JWS are essentially standard Java Applications, far fewer restrictions are imposed on the developer. It is an undemanding task to develop an application which can be distributed as both a standard Java Application and a Web Start. The JNLP protocol also allows for greater freedom in JVM choice, offering the ability to provide automatic upgrades to the user's Java Run-time Environment (JRE), whilst supporting side-by-side installation of multiple JVM versions.

With JWS, the installation of native libraries onto the end user's machine is seamless and eliminates the complexities which can be encountered when attempting the same process using Applets. This is a crucial factor, as LWJGL and consequently jME and Homura rely on native libraries for supporting OpenGL, non-standard input devices such as gamepads and joysticks and also for hardware accelerated audio capabilities provided by libraries such as OpenAL and FMOD.

3.3 Networking for Java-Based Games

Java features an extensive networking API incorporated into the JDK, which many additional networking solutions APIs are built upon, including those designed for Java games. Two such examples of these are Sun's *Project Darkstar* [10] and the jME-affiliated *Java Game Networking* (JGN) [11] which are both open source. Both of these libraries provide game-specific Client-Server APIs, and can be integrated with jME, and therefore Homura. Darkstar provides a shardless Client-Server architecture for MMOG titles using Java, and is capable of supporting thousands of players by utilising the Sun Games Server (SGS) technology as the backbone.

However, as stated in the introduction, the ethos of this project is to create a system that allows a Java-based game to be created with minimal costs, by utilising open-source technologies. Client-Server architectures are popular in game networking, because they are relatively simple to implement, allow the developer to maintain absolute control over the state of game and the use of a centralized authority makes cheat-detection and user access control a simple task. However, scalability is an issue, as it is often inflexible and also extremely costly due to the amount of dedicated hardware required to sustain such design. The network has to be over-provisioned to handle peak loads and limits the deployment of user-generated content [12]. Client-Server games may also have limited life-spans, as they will only remain active whilst the server infrastructure is still supported. The issue of hardware costs means that using Client-Server architecture to support many games is infeasible for small-scale developers, and would create a barrier of entry for using this platform. A Peer-to-Peer (P2P) solution would allow the system to utilize the resources of the connected users (CPU, Memory and Network Bandwidth), creating an easily scalable system and one that is in keeping with the aforementioned ethos. Therefore, the Java version of the JXTA [13] P2P platform (JXSE 2.5) was selected as the underlying networking library for the middleware solution. JXTA is an open source initiative created by Sun Microsystems in 2001 to standardise and provide a set of P2P protocols. JXTA comprises of a set of six protocols that support common P2P networking tasks, such as peer discovery, organization of peers, peer identification and verification and messaging. [14] The JXTA protocols uses the open standards of XML schemas to define their semantics and subsequently allow JXTA to operate independent of language, operating system, network topology and even the underlying transport protocol. The Java JXSE version is the reference implementation of the JXTA standards. JXTA does not preclude the usage of Client-Server topologies, and persistent peers can be utilized to great effect. It neither excludes nor inherently depends on centralized control points. JXTA has been used as the basis for many recent P2P studies, and its feasibility for use within game-related network programming (even large scale applications, e.g. MMOGs) is proven [15].

4. System Overview

The *NetHomura* deployment platform and networking middleware is still evolving. This paper presents the work completed during the preliminary phase of the project. This first stage includes the implementation of the deployment platform, and the initial API for the creation of multiplayer online Homura-Based games. The *Future Developments* section of this paper documents the expectations and upcoming direction of this project, and how it can be further improved and refined in the next phases of development. In this section: we present the combined architecture of the two technologies and the interactions which exist between them; we explain the deployment process, how it can be used to rapidly and securely deliver feature and content rich games via standard web technologies; and we discuss the infrastructure of the network overlay which the client applications will form during the course of a networked game session.

4.1.1 Deployment Platform Overview

The deployment solution provides a web application which provides a portal for access to games built using the Homura platform. The deployment portal provides the following features to the user and developer:

User Accounts/Authenticated Access: Users have their own profile, and are authenticated into the system via an encrypted username/password mechanism. The user's views within the system are customized based on their account details, providing them with an in-game identity, age-restricted access to the games, the ability to create friends with other users, and associates them with an in-game Avatar.
Game Catalogue Facilities: The System catalogues the available games, based on their genre, multiplayer and online facilities, and allows users to easily search and download new games based on their preferences. Previously downloaded games are stored so they can easily start their favourite games, or re-download them to another machine.
Game Statistics Storage: The system stores and displays game statistics, such as the number of times a game has been played, and game high-scores, based on all users across a single game, or all a particular user's score across all games.
Caching and Update Facilities: Once a user has downloaded the game, it is cached so that subsequent executions are instantaneous, and allows updates to be delivered to the user automatically, when they are available.
Logging and Access Control - The System features a logging engine which stores the IP address of the users, any data modifications to the database and allows the system administrator to ban or temporarily disable accounts, add new games to the system as well a system wide configuration which allows the application to be gracefully taken offline for maintenance.

4.1.2 Networking Middleware Overview

The networking middleware provides a high-level implementation of the JXTA networking facilities, allowing the game developer to incorporate networking easily into their application. The middleware currently provides the following features:

- 8-16 Concurrent Players within a single game instance.
- Creation of a network connection from a peer.
- Creation and joining of Peer Groups.
- Sending and Receiving Game-Related Messages.
- Communication of Shared Data to the data repository.
- Representations of users and game session as Java Objects.
- Monitoring of peer latency and bandwidth across the session.

4.2 System Architecture

NetHomura utilises two physical servers to host the solution: a SUSE Linux, Apache 2.0 combined web and application server running PHP 5.2.6 and a Database Server running MySQL 5.0.1. This forms a LAMP (Linux-Apache-MySQL-PHP) configuration, which is commonly used for standard web applications. The application is extremely portable and would work equally well on a Windows-based machine using either Apache 2.0 or IIS. The use of two servers is due to the internal infrastructure available and could easily be combined into a single server. Figure 2

provides an overview of the architecture, illustrating the physical and logical layout of the application and the communications permitted between the server and the game clients.

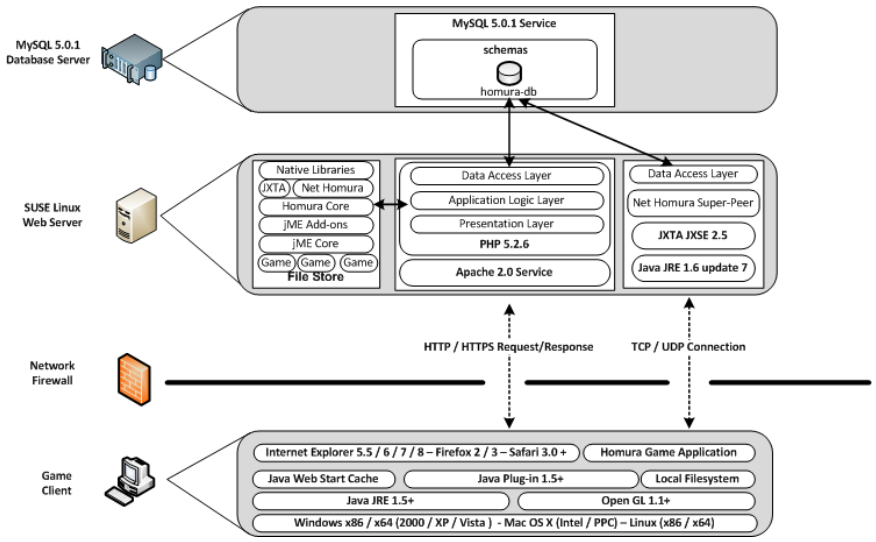

Figure 2: NetHomura System Architecture

The deployment solution is programmed using a blend of Object-Oriented and Page-Oriented PHP, and is separated into a three-tiered architecture:

Data Access Layer – The solution uses MySQL for its persistent storage. A PHP Data Access layer has been implemented, which abstracts the database from the application, allowing the system to retrieve application class instances representing the database objects, and handles the insertion or modification of data within the database based on the current state of the application class. This abstraction also means the database can be replaced with another RDBMS (e.g. Postgres), or use another solution, such as XML files to store the data.

Application Layer – The Application layer provides classes which represent the data in the database and encapsulates the application logic. The system current features implementations of users, games, online game modes; game scores etc. The class instance data is modified by actions carried out by the user via the application interface, using a simple getter/setter pattern.

Presentation Layer – The solution programmatically constructs the HTML and JavaScript-based page template and displays dynamic views of the system to the user, using the classes of the Application Layer. This abstraction means that the entire look and feel of the application can be replaced without affecting the application logic and allows the system to incorporate AJAX/DHTML libraries to enhance the user experience.

The games and supporting libraries (Homura, jME, Native DLLs/SOs) are stored on the server's local file system and are not directly accessible from outside of the server. The files are stored as JAR files and are compressed using the g-zip compression system. This minimizes the size of the game distribution. The partitioning of the

JARs means that, once a client has downloaded their first game the Homura framework is installed, and that any future game installation will only require the game JAR to be distributed, further reducing installation times. The JARs are signed using digital signatures so their integrity and authenticity can be verified, combating binary modification (commonly used for online cheats). This also means that the application does not have to run within the standard Java sandbox, providing additional permissions to the application, such as running Native Libraries (e.g. OpenGL), storing data locally on the user's machine, and creating outbound network connections (used by the middleware). The Server also hosts a persistent JXTA Super Peer, which can be accessed by any of the Clients running NetHomura based games. This Peer is used to advertise the location of other games users, to insert/modify any persistent data (scores, logging information, user-identifiers). This peer is used to allocate network functionality to connected peer applications, delegating more and more power to the peers, and acts as a proxy, so that peers cannot directly access the database. Both the deployment and middleware provide multi-platform support. A game client can currently use Windows, Linux, or Mac OS X Operating Systems and Internet Explorer, Firefox or Safari browsers and requires OpenGL to be installed on the system. The users need at least version 1.5 of the Java JRE (which can be automatically installed by the deployment application). Games are placed in the Java Web Start cache within the local file system.

4.3 Deployment Process

Java Web Start applications are usually deployed using a static JNLP file with a reference link embedded within an HTML page. However, this method is unsecure, and means that anyone with knowledge of the location of the JNLP file can access and deploy the game. Thus, the deployment architecture dynamically creates the JNLP file to protect the games from unrestricted access. Figure 3 illustrates this process for a Windows machine running either Firefox or Internet Explorer.

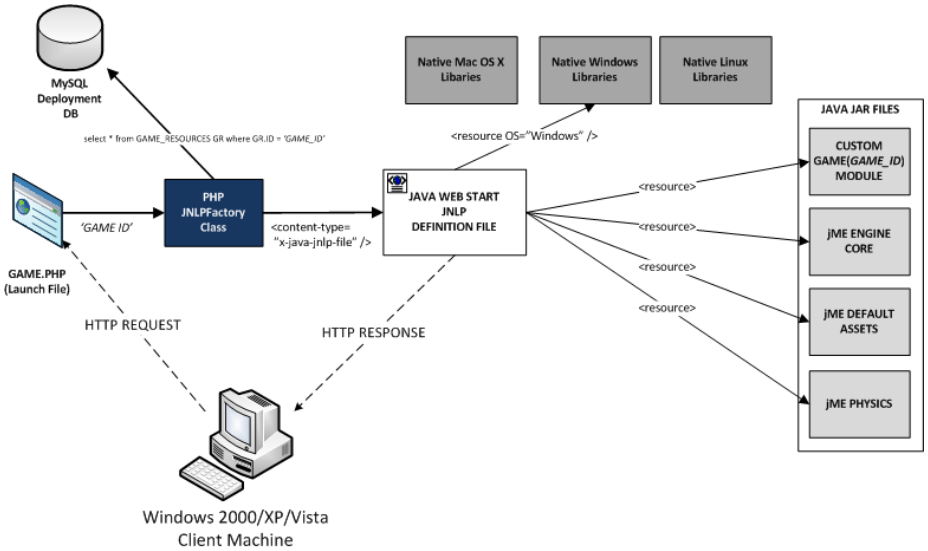

Figure 3: Deployment Process using Windows

A user who is authenticated into the deployment portal can locate a game via the search mechanism, and view the games details (download size, multiplayer modes, online support, screenshots etc.). The application provides a *'Launch'* button, which then invokes the deployment of the game. The game's unique identifier is passed to the JNLPFactory class of the deployment application layer. This class queries the database via the data access layer to retrieve the required resources of the game, JVM settings, properties (used to pass user-specific data to the applications) and dynamically constructs the JNLP file, which is returned to the client via an HTTP response. The browser then passes the JNLP file to the Client's Java Plugin which parses the JNLP file and downloads the game application and resources associated with the application. The Client's architecture and Operating System is established by the Plugin so that only the Native Libraries relevant to the client are downloaded. Once the game is cached, it is then executed using the Client's JVM. The process is similar for subsequent executions. When the JNLP is sent, the contents of the JWS cache on the client's machine is analysed and differentiated against the server copies. If new versions of any of the associated JARs, or any new JARs added to the application (e.g. new level packs, assets) then these are downloaded and returned to the client.

4.4 Peer-to-Peer Networking

The networking middleware of NetHomura utilises the key concepts of the JXTA framework, and builds upon them for the purposes of gaming, with the resulting network created by the framework is a hybrid P2P topology, as illustrated by Figure 4, which details both the network structure and the software architecture of the peers within the system.

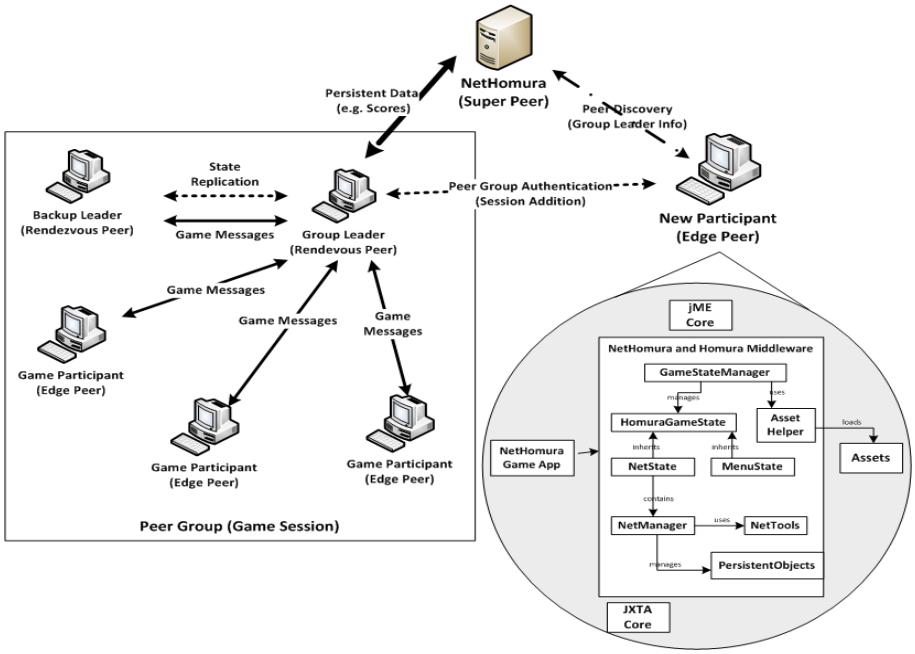

Figure 4: Sample P2P Game Interactions via NetHomura

4.4.1 Middleware Overview

The NetHomura middleware integrates with the Homura Engine to create a *GameStateManager* to control the game. This manages an internal stack of *HomuraGameState* instances. *HomuraGameState* is an abstract class which implements the game loop of each state, providing methods for initialisation of content, handling user input and updating the state of the game world (members of the scenegraph), and a rendering the scenegraph to screen. The NetHomura games are comprised of concrete implementations of this class (e.g. *MainMenuState*, *LoadingState*, *PuzzleGameStat*e, etc.). The middleware provides an additional implementation, *NetState*, which encompasses the additional interactions of a network game, by adding methods to receive messages, send messages, join and leave games. The *NetState* class uses an instance of the middleware's *NetManager* class, which handles peer-management facilities such as discovering available game sessions, creation of new game session, tracking and modifying persistent, shared data objects used within the game, managing references to connected peers, sending messages to particular peers and retrieving messages that are received from peers. The *NetManager* also handles session control, such as disconnecting and joining into both the entire network and game sessions. The role of the game developer using the middleware is to create game-specific messages which inherit from the base *NetHomuraMessage* class. This class encapsulates the in-game messages sent between peers, and using the *NetTools* class to construct efficient managements using the functions to efficiently serialise Java object into messages. These messages can then be broadcast using the *NetManager.* The middleware also provides the concept of *GameSessionAdvertisments,* which are used to create and communicate the details of a particular game session to other peers so that they can participate in a session.

4.4.2 Network Topology

Games built using the NetHomura framework construct a hybrid P2P topology. Initially, the NetHomura *Super Peer* on the deployment server forms a Client-Server network, which acts as the gateway to the P2P game sessions, authenticating the user based on their deployment credentials and transmitting the location of any sessions available for the user's game. A game session is organised as a *Peer group*. The Peer group is assigned a group leader, which becomes a JXTA *Rendezvous Peer*.(initially the creator of the group).The Rendezvous peer is responsible for broadcasting the services offered by the group (e.g. the game session) using JXTA's advertisement framework. The rendezvous peer is also responsible for propagating messages received from other peers within the peer group to all other members [14], who are connected as *Edge Peers*. Edge Peers have a single responsibility, to communicate their current game state to the group leader. A replica of this peer is also created to provide redundancy. If the group leader disconnects, then leadership is switched. Peers are connected using *JXTA Socket* connections, communicating via TCP/UDP. A secure connection using TLS is created between the group leader and the Super Peer to securely transmit persistent data, for entry into the deployment database.

5. Preliminary Analysis

As stressed throughout this paper, the project is in its infancy, and many additional features require implementation and testing in order to construct the final solution.

The deployment system's capabilities were tested on a variety of Virtual Machines, each implementing a different configuration of OS and browser from a clean installation (no previous games) and Java JRE 6 installed. The test machine is a 2.4GHz Intel Core 2 Duo, with 4GB of RAM, an Nvidia 8600GT 256MB graphics card, and connected to the internet via a 2Mb standard broadband connection. The Elemental Puzzle game (developed as part of the Homura Project) is used as the test case. Figure 5 illustrates the Windows Vista deployment of this game using Internet Explorer 7.

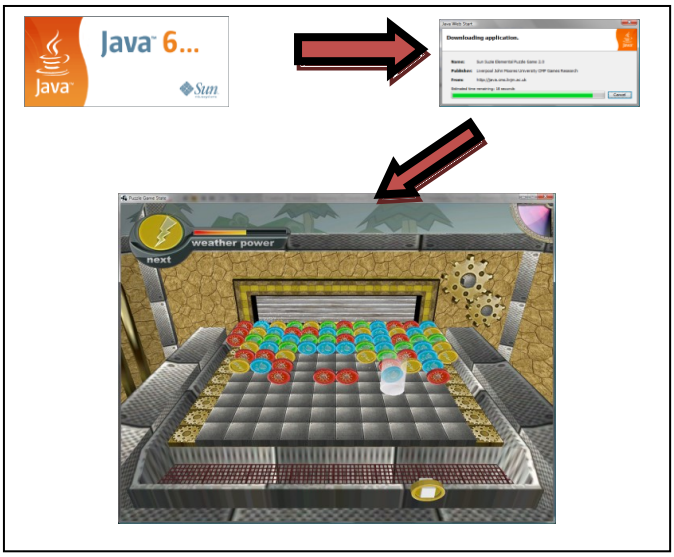

Figure 5: Deployment of Homura Element Puzzle Game

Table 1 illustrates the results of the installation averaged across 10 clean installations of the test machines. The measurement records the time from deployment launch until the game window appears (including JVM invocation time).

Table 1: Homura Game Deployment Tests

Operating System	Browser Tested	Avg. Download Time
Windows Vista	Internet Explorer 7	2m 14s
Windows XP	Internet Explorer 6	1m 57s
Ubuntu 8.04	Firefox 3.02	2m 28s
Fedora Core 7	Firefox 2.04	2m 25s

The game was successfully deployed and executed on each of the test machines. The download also includes the deployment of the entire Homura framework, including jME, so subsequent game installation times will be dramatically reduced. Whilst the raw source of the game is 35MB, the JWS version is 10MB, including the Homura Framework and jME.

Whilst the core of the networking middleware has been created, it is yet to be complete integrated with Homura-based games, and thus cannot be illustrated in use.

6. Future Developments

There are several enhancements and future directions that can be taken in the future development phases in order to fully realize the potential of the proposed solution. Currently, the content of the system is directly added to the database via a set of scripts and stored procedures. The deployment system can be further developed to provide a set of back-end tools which allow the management of games, users, and application configuration through a web interface. This should be a relatively trivial implementation, using the existing data-access and application tiers. The Digital Signature signing process of the game and engine JARs is currently completed manually using the key signing tools distributed with the Java 5 SE JDK. This could be incorporate into either the administration tools of the deployment or the build process of the Homura IDE. The middleware component requires the completion of the test games in order to evaluate its performance and scalability properly. The first phase is to support between 8-16 concurrent players during a game session. The next phase will involve the incorporation of networking algorithms such as dead-reckoning, Area of Interest Management, cheat-detection to increase the scalability of the system by an order of magnitude [16].

7. Conclusion

In this paper, we have presented a novel architecture and prototype system, which aims to unify the deployment of hardware-accelerated Java-based 3D games applications with online capabilities. NetHomura provides a multi-tiered deployment platform that is secure, robust, and easily portable to a wide range of web servers. The networking middleware provided allows developers to built content and feature rich online-games in conjunction with the Homura Engine and IDE. Due to nature of the technologies used within the NetHomura framework, and combined with the Homura engine and IDE, it is possible to enable the creation of a game which, from development through to hosting, deployment and networking, can be created with little or no financial outlay for the developer. This would enable small-scale developers to distribute modern games applications to users worldwide. The games can incorporating online, multi-player game-play utilising the processing and networking resources of the game's users to construct a scalable network providing a smooth consistent and responsive experience. With the future development of this project, it is hoped this goal can be obtained.

References

1. LJMU Homura Sites - Homura Engine and Homura IDE. *Liverpool John Moores University.* [Online] [Cited: October 10, 2008.] http://java.cms.livjm.ac.uk/homura.

2. Pogo Online Game Portal. *EA Pogo.com* . [Online] [Cited: October 10, 2008.] http://www.pogo.com/home/home.do.

3. Flash Based MMOG. *Habbo Hotel* . [Online] [Cited: October 10, 2008.] http://www.habbo.co.uk/.

4. Unity Game Engine. *Unity Game Engine - Official Site.* [Online] [Cited: October 9, 2008.] http://unity3d.com/.

5. Instant Action - Garage Games. *InstantAction.com.* [Online] [Cited: October 10, 2008.] http://www.instantaction.com/.

6. Official Site - Home. *Java Monkey Engine (jME).* [Online] [Cited: October 8, 2008.] http://www.jmonkeyengine.com/.

7. jME Starter Guide - jME 1.0. *Java Monkey Engine.* [Online] [Cited: October 10, 2008.] http://www.jmonkeyengine.com/wiki/doku.php?id=user_s_guide.

8. Official Java Web Start User's Guide. *Sun Microsystems - Java.* [Online] [Cited: October 9, 2008.] http://java.sun.com/javase/6/docs/technotes/guides/javaws/developersguide/contents.html.

9. Java Launching Protocol - JSR Specification Documentation. *Sun Microsystems - Java.* [Online] [Cited: October 10, 2008.] http://jcp.org/en/jsr/detail?id=56.

10. Community Web Site. *Project Darkstar* . [Online] [Cited: October 8, 2008.] http://www.projectdarkstar.com/.

11. JGN - Java Game Networking API - Official Site. *Java Game Networking.* [Online] [Cited: October 10, 2008.] http://javagn.org/.

12. Knutsson B., Lu H., Xu W., Hopkins B. *Peer-to-Peer Support for Massively Multiplayer Online Games.* s.l. : INFOCOM, 2004. Vol. 1.

13. Official JXTA Development Site. *Java Development Community - JXTA.* [Online] [Cited: October 10, 2008.] https://jxta.dev.java.net/.

14. J., Verstrynge. *Practical JXTA.* s.l. : Lulu Enterprises, 2008.

15. El Saddik A., Dufour A. *Peer-to-Peer Suitability for Collaborative Multiplayer Games.* s.l. : Proceedings of the Seventh IEEE International Symposium on Distributed Simulation and Real-Time Applications, 2003.

16. Smed J., Hakonen H., Koukoranta T. *A Review on Networking and Multiplayer Computer Games.* s.l. : Technical Report, Turku Centre For Computer Science, 2004.