Muddleware for Prototyping Mixed Reality Multiuser Games

Daniel Wagner[*]
Graz University of Technology

Dieter Schmalstieg[o]
Graz University of Technology

Abstract

We present Muddleware, a communication platform designed for mixed reality multi-user games for mobile, lightweight clients. An approach inspired by Tuplespaces, which provides decoupling of sender and receiver is used to address the requirements of a potentially large number of mobile clients. A hierarchical database built on XML technology allows convenient prototyping and simple, yet powerful queries. Server side-extensions address persistence and autonomous behaviors through hierarchical state machines. The architecture has been tested with a number of multi-user games and is also used for non-entertainment applications.

Keywords: mixed reality games, client-server architectures, middleware

Index Terms: C.2.4 [Computer Communication Networks]: Distributed Systems - Client/server, H.5.1 [Information Interfaces and Presentation]: Multimedia Information Systems - Artificial, augmented, and virtual realities

1 Introduction

As mobile gaming technology is becoming increasingly available, shared game spaces that were previously bound to desktop computers within a fixed infrastructure are now moving to handheld game consoles and cell phones. Mobile gaming is not simply a trend towards a more convenient hardware platform, but a fundamental paradigm shift in computer gaming. The resulting *Mixed Reality Games* [1] bring the gameplay out of the virtual world and back into the real world. The requirements for mixed reality (MR) games are inherently different from conventional online games. Mixed reality games impose a number of unique challenges which differ significantly from conventional games:

Communication: When relying on wireless communication such as WiFi or GPRS, loosing network connection during gameplay is not an exception but rather a rule. Users, thus must be able to continue playing despite being temporarily disconnected [2].

Heterogeneity: Making the best use of game client diversity is an important aspect of MR game design and not an unwanted side effect. For example, users of mobile and stationary clients can exploit their complementary capabilities for collaborative problem solving. To allow efficient development, game software must therefore exceed conventional standards concerning modularity, lightweight footprint and cross-platform availability. A low-end client must be able to participate in the overall game with only a minimum of required software.

Persistence: Like online role-playing games, MR games are situated in a persistently evolving universe – however, in case of MR, this is the real world. Since predictable behavior or even reachability of clients in the real world cannot be expected, all important game data must be stored persistently.

Rapid prototyping: To facilitate the experimental user-centered game design necessary for MR game development, the network communication between game components should be accessible via loosely typed, interpreted scripting in a way that works similar to game behaviors scripting.

To address these specific needs for MR games middleware, we have developed a communication framework called *Muddleware*. It is loosely inspired by Tuplespaces [3] – the name Muddleware refers to a complex "pile" of data tuples. Muddleware extends the traditional Tuplespace idea with concepts from online games, multiuser virtual environments and web-based information systems. To our knowledge it is the first communication platform to combine content-based addressing, open standards (XML), server-side behaviors and support for small handheld devices. This paper presents the design rationale and implementation approach and showcases some MR application examples.

2 Related Work

Muddleware is inspired by the concept of a Tuplespace (also often called Blackboard), an associative memory that allows content based addressing. There are modern implementations such as IBM TSpaces [4] and JavaSpaces [5], and also variants for mobile environments such as LIME [6] which transiently shares Tuplespaces among multiple clients. The transient sharing makes LIME very dynamic, but is inappropriate for deterministic gameplay since the availability of essential game data cannot be guaranteed.

We found the Event Heap [7] designed for synchronous groupware, very influential for our work. The Event Heap among others introduces a publish-subscribe mechanism for ordered communication. Similarly, the Enchantment Whiteboard [8] is used to exchange data between various wearable components in a Body Area Network through a subscription mechanism.

Unreliable wireless network connections for mobile MR was specifically addressed in [9] using replicated databases and fault-tolerant communication.

Finally, the EQUIP platform [10], designed to support ubiquitous computing applications, is similar in its goals and methods to our work. However, our choice of XML and XPath for data representation and queries is more open and expressive than previous approaches.

3 System Architecture

Another important requirement in designing Muddleware was a platform-agnostic approach which allows for a wide variety of target devices and the use of open standards Multiuser games often use SQL for storage, but in contrast to these games, which have very well defined client infrastructure and gameplay, MR games require support for heterogeneity and prototyping not provided by SQL but well supported by Tuplespace-like models.

3.1 Built on XML Technology

The *Extensible Markup Language*, XML is a widely used network-centric technology. XML elements with named attributes

[*]e-mail: wagner@icg.tu-graz.ac.at
[o]e-mail: schmalstieg@icg.tu-graz.ac.at

IEEE Virtual Reality Conference 2007
March 10 - 14, Charlotte, North Carolina, USA
1-4244-0906-3/07/$20.00 ©2007 IEEE

can be seen as a specific representation of a tuple. However, the XML Document Object Model (DOM) uses a hierarchical data model, which permits a recursive definition of tuples. Muddleware was thus designed to adopt XML technologies for data storage, addressing and retrieval of data.

Among the important advantages that led to the adoption of XML are its human readable representation, its aptitude for quickly creating and changing structured data collections (lists, trees, records), automated validation and wide support by efficient open source tools and libraries. Above all, we found XPath[1] to be very suitable as a query language that significantly surpasses conventional template matching of (flat) tuplespaces in expressive power, yet is still very efficient and fast.

3.2 Database Server

The core of Muddleware is a memory mapped database that provides persistence and can be addressed associatively using XPath. Clients connect to the server (see Figure 1) by any of four APIs: Immediate C++, Shared Memory C++, Java and Muddleware Script.

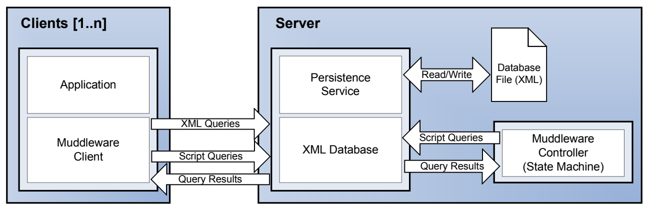


Figure 1: Core Muddleware components

All data elements are stored as nodes of a modified TinyXML[2] DOM. Clients store arbitrary messages as XML fragments, and use XPath to specify query or update operations. Muddleware also supports atomic conditional operations, essentially implementing a lightweight transaction protocol. Operations chaining allows subsequent operations to be executed only if preceding operations were positively evaluated.

In addition to immediate databases operations, Muddleware allows clients to register Watchdogs (observers) for updates: as soon as an observed node changes, the client is informed about the update. This removes the need for polling and provides a simple, yet powerful publish/subscribe mechanism that can be used to create specific communication channels between clients.

Muddleware's handling of concurrent network sessions is implemented using the ACE[3] ProActor pattern. At short intervals a background thread dumps the complete database to disk; we plan to upgrade this to a professional database system.

3.3 Queries with XPath and Muddleware Script

XPath is a simple language for addressing elements and attributes in a hierarchical DOM. It allows template matching by specifying mandatory values for node types and elements to be matched, and allows navigation of the hierarchical structure. An XPath query will return zero or more XML elements, but does not allow iteration in the navigation. We found this to be a suitable compromise for content-based retrieval as necessary for MR games. The hierarchical structure of the DOM is typically used to represent a geometric or logical hierarchy of game entities.

Muddleware Script is a simple XML dialect for expressing queries. Clients can create scripts and register these at the server. Each script is precompiled into a tokenized form for fast execution and stored for future invocations. The scripting language allows invoking all database actions such as adding, removing and querying elements and attributes. Multiple actions can be hierarchically combined using boolean operations and functional composition – results of one query can be used as input for another query. Results are passed on from deeper nested actions to their respective parent nodes in the XML graph. An XML schema helps developers to ensure validity of script files.

Figure 3 shows a short sample action that checks if a non-player character was defeated. The sample also demonstrates the strength of XPath's associative addressing: Qualifiers for attributes can be specified along the path through the DOM graph down to the target item. In this example, first a database action is executed that reads an attribute from a node addressed via the specified XPath. The result is passed on to the parent action which in this case compares it to a specific value. Only if the attribute called 'state' is set to 'dead' the action will return true.

3.4 Muddleware Controller

A typical MR game needs to advance even if the players are inactive. For example, it may be necessary to enforce time-outs. This requirement is addressed by a server-side component, the Muddleware controller. Its task is to observe the database and react to changes based on internal interpretation of custom logic by writing new data into the database. Its internal logic is built on a finite state machine, such as frequently used in nonlinear story-driven games.

Each path in the graph starts at a specific entry state. Actions, expressed as Muddleware scripts, are executed upon entry and exit of states. Each state can have one or more transitions to other states. A transition is only taken if the transition's guard condition holds. Transition guards refer to conditions on elements in the XML database, updated by Muddleware script actions. The controller writes its own state into the XML database, which allows monitoring and influencing the state machine remotely.

The controller's state graph is supplied as an UML state chart, encoded as an XMI file. The XML dialect XMI can easily be parsed in an XML framework and the controller can directly operate on the resulting DOM. This type of state chart was originally designed for software engineering, and high quality graphical editors exist for visual programming.

3.5 Muddleware Client

Muddleware provides a choice of four client-side APIs:

C++ Immediate Client API. The C++ immediate API provides lowest-level access to client-side Muddleware. The single operation API can invoke simple methods such as getElement(). The disadvantage of this simple API is the use of blocking I/O. In practice this API is used to respond to infrequent user-triggered events.

```
Request* request = Request::create();
Reply* reply = Reply::create();

request->updateAttribute("/MyApp/Owner/@name", "Peter");
request->removeAttribute("/MyApp/Owner/@address");
request->addElement("/MyApp/User",
   "<User name='John' age='35' />");
request->removeElement("/MyApp/User[name='Fritz']");

connection->sendAndReceive(*request, *reply);
```

Figure 2: Code excerpt using the multi operation API to perform 4 operations at once

[1] http://www.w3.org/TR/xpath.html
[2] http://sourceforge.net/projects/tinyxml
[3] ACE toolkit: http://www.cs.wustl.edu/~schmidt/ACE.html

```
<ActionEqual name="checkBossEnemyDefeated" value="dead">
  <ActionExecute operation="getAttribute" xpath="/Games/Game[@name='Dungeon']/Enemy[@name='Dragon']/@state"/>
</ActionEqual>
```

Figure 3: Sample Muddleware script code sequence

The multi operation API stores operations in a request object which is sent to the server on demand. Multiple operations can be created ahead of time and in batches. Moreover, the communication occurs asynchronously in a separate thread. Figure 2 shows an example of the multi operation API. All four operations are stored in the request object and then sent together to the server.

C++ Shared Memory API. Manually sending updates and receiving query results is often tedious in application level code. Although it provides finest graded control, such a level of control is usually not required. Remote object libraries are more convenient since they allow accessing server data via proxy objects that behave like local data objects. Even when sharing data is not the goal such a mechanism can be used for persistence, which is especially interesting for clients that have no storage capabilities or lack required robustness.

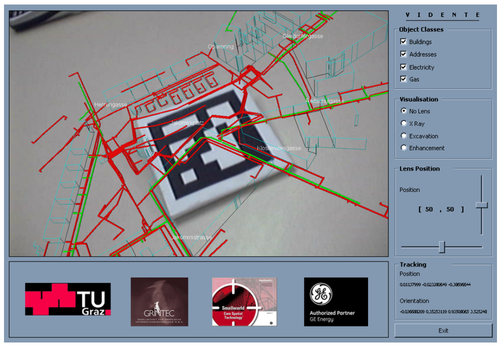
The C++ shared memory API implements such a feature set and is therefore the preferred C++ client API for most applications. A programmer can declare a C++ data structure directly in C++ source, which is then automatically implemented by usage of macros and internal tool methods. No interface compiler such as used by CORBA is required.

Java Client API. A pure Java implementation of the immediate API was developed for using Muddleware with Java applications such as browser games and Java-enabled mobile phones.

XML Client API. Instead of using the C++ or Java client API to batch database operations, client developers can also write Muddleware script. A script file is loaded by the client API, sent to the server which registers all embedded actions. Clients can then invoke an arbitrary number of operations with a single call to a registered XML action. This technique reduces the network load and provides a more data-driven approach than writing C++ code.

3.6 Graphical User Interface Generation

A recent extension to Muddleware allows the generation of graphical user interfaces using Qt Designer[4] without any programming. Thekla automatically pushes the state of all Qt widgets to the Muddleware server, which forwards the state to the application(s) via a Watchdog notification.



Object Classes
[✓] Buildings
[✓] Addresses
[✓] Electricity
[✓] Gas

Visualisation
(●) No Lens
( ) X Ray
( ) Excavation
( ) Enhancement

Lens Position
Position [50 , 50]

Tracking
Position
0.01137999 -0.023280649 -0.38695544
Orientation
-0.036588209 0.35253119 0.93508065 3.525248

Exit

Figure 4: A Qt GUI controlling an Open Inventor application using Thekla

A special extension for the Open Inventor scene graph library allows creating connections between scene graph attributes ("fields") and Qt widgets without writing a single line of source code. Figure 4 shows a simple Qt user interface controlling an OpenInventor application with a potentially remote Qt user interface.

3.7 Performance

XML encoding and decoding as well as XPath decoding is known to be a computationally intensive task. To prove that the Muddleware server is capable of serving many clients we performed some benchmarks on a Windows XP machine with 1 GB of RAM and a Pentium 4 processor at 3.0 GHz:

A simple benchmark showed that the XML server can easily handle thousands of complex requests per second. To test Muddleware with a more practical example we measured the server's CPU and network load while running 50 instances of a pre-recorded 4-player Virtuoso game (see below) in parallel. The server's CPU load with 200 concurrent clients is ~60% while the 100 Megabit network was operating at a capacity of just 4%.

4 Applications

The system presented in this paper is beyond concept stage and was used in several games and other projects that are described in more detail below. With the availability of Muddleware, students and researchers at our lab were able to spend more time on creating compelling games and AR/VR software than writing multi-user servers again and again. Before, our games usually included custom designed and implemented game servers.

4.1 Virtuoso – History of Arts

Virtuoso [11] was the first game to use Muddleware. Virtuoso is a collaborative educational Augmented Reality game running on off-the-shelf Windows Mobile PDAs. The game's objective is to sort a collection of artworks according to their date of creation along a timeline (see Figure 5a) drawn on wall-mounted billboards (left = earlier, right = later). Every marker (fiducial) on the timeline carries one of the artworks (see Figure 5b), which are only visible through the player's PDA. Initially the artworks are in random order. Players can pick up any artwork with their PDAs, by clicking on the artwork on the display and drop it on a free position by clicking on the free fiducial on the display.

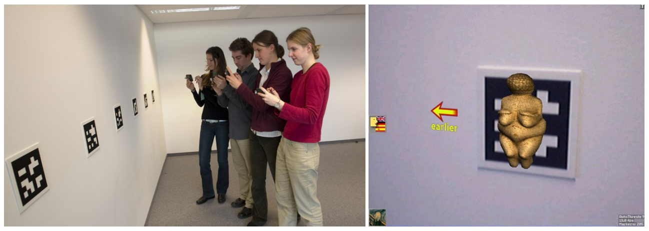


Figure 5: Virtuoso game: a) 4 players in front of the timeline. b) virtual art item on the timeline visible through the PDA.

The game features a set of 20 artworks. Before the game starts the game master selects an arbitrary subset by using a graphical tool that updates the selected subset onto the XML server. All client devices are connected to the server via a wireless network (WiFi). At startup they retrieve the list of available artworks and register Watchdogs for those XML elements that represent the game state. Consequently, whenever a player takes an artwork off

[4] Qt: http://www.trolltech.com/products/qt

the wall onto her PDA it immediately vanishes from the wall on the screen of all other players too.

4.2 Enigma Rally

Enigma Rally is a situated, mixed reality multi-player game that is tightly integrated into a real museum environment. Enigma Rally uses all currently available features of Muddleware.

The players gather around a checkpoint terminal where a PC runs a map of the exhibition and shows the state of all game hotspots. Each player receives a PDA to interact with specific museum exhibits that were enhanced using AR and are thus part of the game. This allows reviving historical exhibits that could otherwise not be operated anymore. E.g. players can use an original world war II radio finder to track and record a radio signal in a similar way it was done at that time. The exhibition includes several PC-controlled stationary hands-on exhibits such as a real Morse button connected to a PC that are also interfaced into the game. Figure 6 shows how the PDAs, the check-point terminal and the hands-on exhibits are integrated using Muddleware.

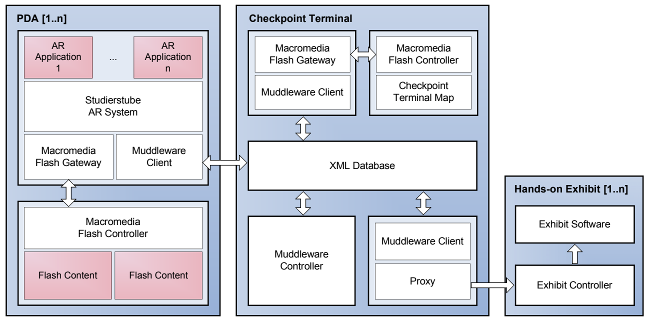


Figure 6: Infrastructure of the Enigma Rally game

The team's checkpoint terminal is not only used for showing the map and the team statistics but also runs the XML database and Muddleware controller which together make up the game server. The controller reacts on specific database events such as unlocking new hotspots as soon as previous ones were finished. The terminal's screen is driven by a Macromedia Flash application that communicates with the server via a Flash-Muddleware gateway module.

4.3 Augmented Reality Agents

The AR Agents project [12] aims at displaying animated agents in the real environment for story telling and as personal assistants. These agents are embodied by real and virtual objects and can appear on a large number of devices ranging from PDAs over desktop screens and HMDs to large projections walls. Agents can migrate from one device to another autonomously, always searching for the best means to fulfill their tasks. Due to the distributed nature of the project, there is a strong need for a flexible infrastructure which naturally lends itself to using Muddleware. Muddleware's XPath ideally complements the hierarchical AR Agents “brain” structure. Using XPath queries agents can quickly identify new habitats to migrate to that match their requirements (display size, processing power, etc.) and inform all participating devices about these changes.

4.4 GeneView

After a first success with AR games, Muddleware is now also used for non-game setups. The GeneView[5] project aims at giving researchers a comfortable and highly customizable tool for browsing, brushing and viewing huge genetic datasets. It is implemented as a distributed Java application that spans its user interface onto an arbitrary number of screens driven by a set of networked PCs. GeneView uses Muddleware for distributing configuration data as well as sharing commands generated by user events. Commands are sent to the XML database and are therefore available to all stations which allows controlling all displays from all PCs by multiple users concurrently.

5 Conclusions and Future Work

Muddleware is a communication platform for mixed-reality multi-user games that is light-weight and highly portable. It has proved its applicability in several MR game projects. In the future we plan to extend the Muddleware server with loadable modules that can feed data from external data sources into the database. Furthermore, we plan to extend the Muddleware controller to directly execute C++ code for transition guards.

6 Acknowledgements

This project was funded in part by Austrian Science Fund FWF under contract No. L32-N04 and Y193. The authors want to thank M. Kalkusch, C. Pirchheim and I. Barakonyi for their help.

References

[1] Stapleton, C. B., Hughes, C. E., Moshell, J. M., Mixed fantasy: Exhibition of entertainment research for mixed reality. International Symposium on Mixed and Augmented Reality (ISMAR), pp. 354–355, Tokyo, Japan, 2003

[2] Barkhuus, L., Chalmers, M., Tennent, P., Hall, M., Bell, M., Sherwood, S., Brown B., Picking Pockets on the Lawn: The Development of Tactics and Strategies in a Mobile Game, UbiComp 2005, Tokyo, Japan, 2005

[3] Gelernter, D., Generative communication in Linda, ACM Transactions on Programming Languages and Systems (TOPLAS), Vol. 9(1), pp. 80-110, 1985

[4] IBM TSpaces, http://www.almaden.ibm.com/cs/TSpaces, 2000

[5] Freeman, E., Hupfer, S., Arnold, K., JavaSpaces Principles, Patterns, and Practice, Pearson Education, ISBN 0201309556, 1999

[6] Murphy, A.L, Picco, G.P., Roman, G, LIME: A Middleware for Physical and Logical Mobility, International Conference on Distributed Computing Systems (ICDCS), pp. 524-541, 2001

[7] Johanson, B., Fox, A., Hanrahan, P., Winograd, T., The Event Heap: A Coordination Infrastructure for Interactive Workspaces, IEEE Workshop on Mobile Computing Systems and Applications (WMCSA), pp. 83-93, 2002

[8] DeVaul, R., Sung, M., Gips, J., Pentland, A., MIThril 2003: Applications and Architecture, International Symposium on Wearable Computers (ISWC), 2003

[9] Brown, D. G., Julier, S. J., Baillot, Y., Livingston, M. A., Rosenblum, L. J., Event-based data distribution for mobile augmented reality and virtual environments, Presence - Teleoperators and Virtual Environments, Vol. 13(2), pp. 211-221, April 2004

[10] Greenhalgh C., Izadi S., Rodden T., Benford S.: The EQUIP Platform: Bringing Together Physical and Virtual Worlds. Technical Report, University of Nottingham, 2001. http://www.crg.cs.nott.ac.uk/~cmg/Equator/Downloads/docs/equip-platform.pdf

[11] Wagner, D., Billinghurst, M., Schmalstieg, D., How Real Should Virtual Characters Be?, SIGCHI Conf. on Advances in Computer Entertainment Technology 2006 (ACE 2006), LA, USA, 2006

[12] Barakonyi, I., Schmalstieg, D., Ubiquitous Animated Agents for Augmented Reality, 5th International Symposium on Mixed and Augmented Reality (ISMAR'06), Santa Barbara, CA, USA, Oct. 2006

[5] http://www.icg.tu-graz.ac.at/research/CGIS/GENVIEW