Adaptable Client-Server Architecture for Mobile Multiplayer Games

Abdul Malik Khan, Ivica Arsov,
Marius Preda, Sophie Chabridon
Institut TELECOM
TELECOM SudParis
9 rue Charles Fourier
91011 Evry cedex, France
CNRS UMR SAMOVAR
{Abdul_malik.Khan, Ivica.Arsov,
Marius.Preda, Sophie.Chabridon}@institut-telecom.fr

Antoine Beugnard
Institut TELECOM
TELECOM Bretagne
Computer Science Department
CS83818
F-29238 Brest cedex 3, France
Antoine.Beugnard@institut-telecom.fr

ABSTRACT
Most of the multiplayer games available online are based on a client-server architecture because this architecture gives better administration control to the game providers. Besides controlling the account and payment information of the players, this architecture also prevents players from cheating as all the game logic is executing on the centralized server. We proposed a server assisted approach for mobile games in [2]. However, because of the varying and high latency of wireless networks and of the changing consistency requirements during the game play, it is difficult to keep the user experience highly interactive in client-server architecture. This paper presents an adaptive hybrid client-server architecture which changes its behavior according to network and game environment variations to improve game state consistency across different mobile terminals. The server applies consistency mechanism on its side, as in the traditional client-server architecture and dynamically switches to apply a client side consistency mechanism when inconsistencies occur at the client side because of the change in network conditions and/or game requirements. We have evaluated our approach on a car racing game. The results show that we can obtain an improved global consistency under a high and varying latency network using our dynamically adaptable approach.

Categories and Subject Descriptors
SD.1.6.8 [Distributed Systems]: Simulation and Modelling.

General Terms
Client Server Game Design and Algorithms

Keywords
Multiplayer Mobile Games Architecture, Latency Hiding, Consistency Algorithm, MPEG-4

DISIO 2010 March 15, Torremolinos, Malaga, Spain. Copyright 2010 ICST, ISBN 78-963-9799-87-5.

1. INTRODUCTION
In client-server architecture, the server is responsible for all the communications between the connected clients. In case of a game server, it is the server which hosts all the game execution states; the client is only responsible for displaying the game's virtual world. The weakness of this architecture is its bandwidth requirement and scalability. Peer-to-Peer architectures have then been proposed to solve these issues. The major problem with such architectures is that each player is having a complete game state running on their terminal which is open to cheating from the player. Another issue is that in the case of a mobile terminal, we have a variety of heterogeneous devices having different limited processing and memory capacities, limited screen resolution and different operating systems. So it becomes difficult for a game developer to write a game code which could run on a multitude of these terminals. Also, with mobile phones using 3G networks, P2P is not very common because IP multicast for mobile phones is yet to be widely established, although in future they may become quite common [3]. For these reasons, client-server architecture is favored. Moreover, a centralized control is an important part in making the game profitable for game companies. The game administrators can charge subscription fees from the players hence compensating for the game development costs. Apart from this, a centralized game is easier to implement as there is no need for peer-discovery, distributed event ordering, cheat prevention or distributed storage management as in a peer-to-peer architecture.

When mobile clients are connected to a server through a wireless network, the network delays can vary considerably and jitters can occur. As the game logic executes on the server which normally has some game consistency maintenance and event ordering mechanisms, the update events can reach different players at different points in time because of the variation of the network latency. This can cause inconsistencies between different clients. Also, the server must send the update messages frequently enough so that a consistent virtual world can be displayed on the client's mobile terminal. This can increase the bandwidth requirement of the server considerably, hence making it less scalable.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8704
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8704

In this paper, we propose a hybrid client-server approach in which the game state is partially shared by the mobile terminal and which adapts itself on the fly to the varying network latency and changing game environment to achieve consistency among different players. This adaptation takes place at run time and the system dynamically decides whether a consistency mechanism is required on the mobile client's side or not. We believe that this architecture increases the consistency of the game state on the client side in case of high latency, as in a 3G wireless network. Apart from maintaining better state consistency, this approach relieves the game server from sending update messages too frequently in case of a heavy network load, thus making it more scalable. Like a peer-to-peer architecture, this approach uses client-side capacities to reduce bandwidth requirements for the server and improves consistency in high latency wireless networks. However, we do not need a complete game logic on mobile terminals if they have limited processing and memory capacities or if the network latency is not very high.

This paper is organized as follows. In section 2, we analyze some related works concerning consistency maintenance in multiplayer games and 3D visualization on mobile phones. In section 3, we first discuss server-oriented or thin-client architectures in which all the game is executed on the server side while the clients are responsible only for rendering the game state; we then discuss a heavy-client approach, in which servers are used mainly for message passing and player's account storage. Then, in section 4, we present our hybrid architecture and a communication protocol, and also give an evaluation of our proposal. In section 5, we conclude our paper and discuss some future work.

2. RELATED WORK

In [5], the authors presents a consistency mechanism for mirrored-server architecture in which the servers are connected in peer-to-peer while clients are connected to the nearest server in a client-server manner. The idea is to reduce the bandwidth requirement on the servers by having multiple servers and to reduce the latency between clients and the servers as each client has to connect to the geographically nearest server. For consistency maintenance, a trailing state synchronization (TSS) algorithm is proposed. TSS keeps two copies of the same game world, each at a different local simulation time separated by some synchronization delay. The latest one in time is called the leading state and the other one, the trailing state. When an inconsistency is detected in the leading state and rollback is required, TSS copies the game state from the trailing state to the leading state, and then performs all commands between the inconsistency point and the present point again.

This architecture is a static design time approach. Moreover because of the issue mobility, in case the clients are mobile terminals, it suffers from dealing with variations in latency which may cause huge inconsistencies at each client and could require lots of roll-backs. Besides, roll-backs require storing previous states in memory, which is limited in case of mobile devices. This algorithm also considers that all the clients are equivalent in terms of their capacities. In our approach, we decide at run time whether a client-side consistency is required or not keeping in view variations in the network and game environment and also dealing appropriately with heterogeneous devices having different capacities.

In [4] Chia-chun Hsu et al. propose to use a cluster of servers for multiplayer games. Their idea is to divide games into different regions, each server executing different regions of the game. This increases the scalability and reduces the bandwidth requirement on the server side. However, again this is a static architecture which suffers from client side latency issues because of changing network latency and jitters. Also, as in [5], it does not take into account the heterogeneity of clients' devices.

MACVE [14] is a *Mobile Agents* based frame work to dynamically change network architecture to make it more scalable. "Each mobile agent is a software component which assumes an independent task to provide a certain service for the system". Agents collaborate between them to maintain the distributed system. Agents do not bond to any fixed host and can migrate between hosts to provide services. This approach is for IP based networks and does not treat consistency issues.

Apart from the hybrid architectures that we discussed above, the thin-client approach has also been used for visualizing 2D and 3D data. It is appropriate for accessing huge amount of data [7], for example medical data, 3D scanned data or very complex CAD models from less powerful devices like mobile phones or PDAs. The techniques for thin client access can be separated in two main groups: pixel-based and graphics primitives based. The pixel based techniques capture the rendered results and send it to the client. Some of them use video streaming, as the system proposed by Sanna A. in [13], while others use image based transfer [15]. The graphics primitives based techniques transfer the data to the client, therefore the client has to be capable of rendering the data. These kinds of architectures are closely connected to ours. The type of data transferred depends upon the application itself and on the capabilities of the terminal. For example, the application uses 3D objects. If the client cannot render 3D objects, they can be converted to 2D lines and sent to the client. This architecture was proposed by Diepstraten J. [16]. If the client is capable of rendering 3D graphics, the 3D objects can be sent with some adaptation [17].

3. CLIENT-SERVER ARCHITECTURE FOR MOBILE GAMES

In this section we discuss the server centric (also called thin client) and heavy-client approaches and their limitations in mobile setting.

3.1 Server Centric Approach

Two main categories of requirements drive our developments in proposing the client-server architecture:

- For game creators, the deployment of a game on a large category of terminals should not conduct to additional development costs,

- For players, the game experience (mainly measured in the game reactivity and loading times) should be similar or better compared with a locally installed and executed game.

The main idea is to separate the different components presented in a traditional game into components that are executed on the server and components that are executed on the client terminal.

As demonstrated in [13], the MPEG-4 standard provides technologies able to represent (in a compressed form) a scene graph and graphics primitives. A MPEG-4 player is then able to

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8704
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8704

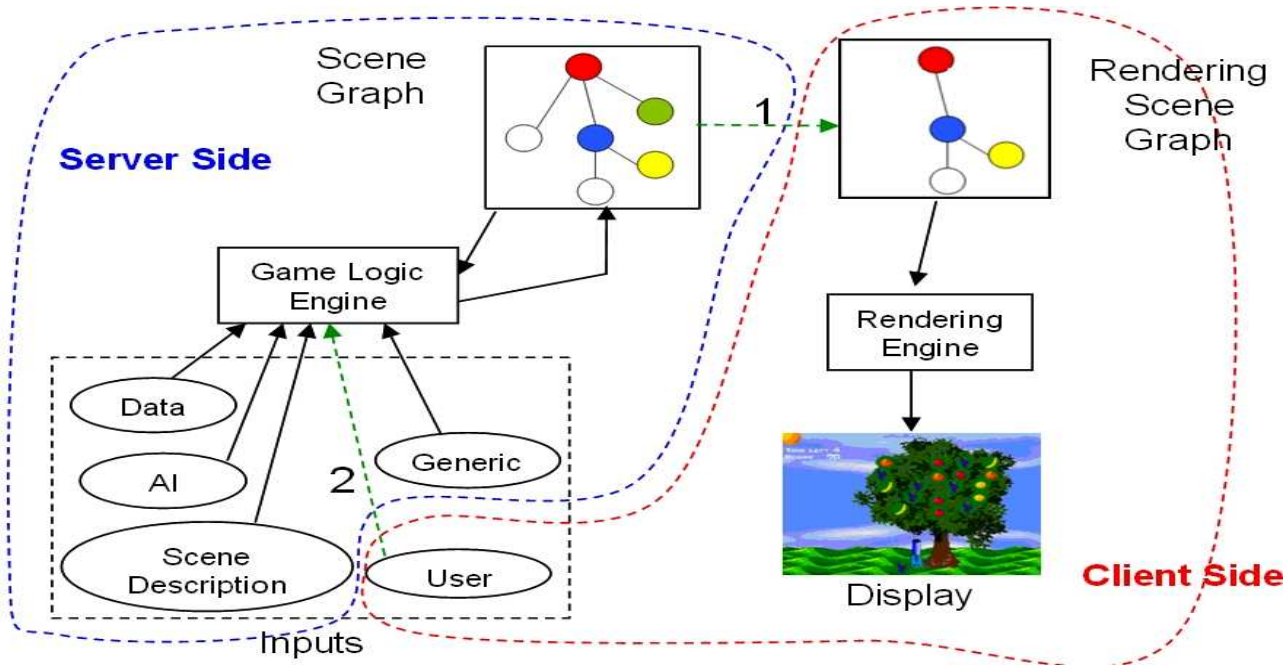

Figure 1. Proposed Architecture for Mobile Games Using a MPEG-4 Player on the Client Side.

interpret them to produce a synthetic image. It was proposed to replace the rendering engine of the game with a MPEG-4 player, with the following consequences: the scene graph (or parts of it) has to be transmitted to the client and the user input (captured by the client) has to be transmitted to the server. Figure 1 illustrates the proposed architecture.

The main underlying idea of the architecture proposed in Figure 1 is to execute the game logic on the server and the rendering on the terminal. In addition, the player receives only what is necessary at each step of the game (interface 1). For example, in the initial phase, only some 2D graphics primitives representing the menu of the game are transmitted. When the game starts, the 3D assets are sent only when they are used, the MPEG-4 compression ensuring fast transmission. During the game play, the majority of the communication data consists in updates (position, orientation) of assets in the local scene. We should note that for games containing complex assets it is also possible to download the scene graph, i.e. an MPEG-4 file, before starting to play. The off-line content transferring has similar functionality as the caching mechanism proposed in [6]. In addition, it is possible to adapt the graphics assets for a specific terminal [11] allowing for the best possible trade-off between performance and quality. In the proposed architecture, the communication characterized by interfaces 1 and 2 in Figure 1, unlike in [12], is based on a higher level of control: the graphic primitives can be grouped and controlled as a whole by using few parameters. The MPEG-4 standard allows any part of the scene to be loaded, changed, reorganized or deleted. For example, the same game can be played on a rendering engine that supports 3D primitives, most probably accelerated by dedicated hardware, and simultaneously on a rendering engine that only supports 2D primitives. This flexible approach allows the distribution of the games on multiple platforms without the need to change the code of the game logic. Another advantage is the possibility to improve the game logic without additional costs for the client, allowing easy bug-fixing, adding features and different optimizations.

In the case of multiplayer gaming, synchronization between different players is directly ensured by the server by controlling at each step the scene graph of each terminal. It means that all players will always see the game in the same state. Therefore, the use of techniques for synchronization between the clients is not needed. The main drawback of the proposed method is the sensibility to the network latency. Big latencies can cause different players to view the game in different states at the same global time. This can cause disadvantage for the players that see the state later than the other players. To solve this synchronization problem, it is necessary to have a thin layer of the game logic on the client side itself for consistency maintenance. In the next section, we discuss client side consistency maintenance approach.

3.2 Client-Centric Approach

In this section, we discuss how consistency is maintained in a client-server architecture in case of a heavy client (client-centric approach) when the logic of the game resides on the client side such as in [18]. In this case, the game logic is totally on the client side and the server performs only message passing and some administrative works such as maintaining a database of players, their accounts etc.

In this paper we consider the clients to be on mobile devices such as mobile phones, PDAs, and laptops. The client-centric or heavy-client approach, as we call it, gives the players more control over the game and reduces the bandwidth requirements on the server side. This approach has the following advantages over the server-centric approach:

- With the deployment of all the game logic on the client side, we do not need high capacity servers or mirrored servers dedicated to execute the game logic [5].

- Because all the logic is on the client side, the clients now need to send messages to other clients through the server less frequently and only when required, using dead-reckoning

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8704
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8704

algorithms [1]. This resolves the bandwidth issue which could be a bottleneck in case the server has to send messages at a frame rate for managing the display on the client side.

This approach has some disadvantages:

- Because of the heterogeneity of mobile devices, it is difficult to develop a game that runs on so many different devices which have different memory capacities, different screen resolutions and are connected to the server through different networks such as WIFI, Bluetooth and/or GPRS.

-Because of the player's complete control over the game, he/she can add cheating mechanism thereby changing the end game results.

-As the game logic resides on the client side, the total delay is equivalent to client-to-client delay which could be approximately double that of server-to-client delay. In case of high network latency, the resolution of state inconsistencies becomes even more difficult.

We have previously proposed a dynamic consistency maintenance mechanism for heavy-client-light-server architecture [8], where we combine two different synchronization schemes namely dead-reckoning [1] and local-lag [10]. In case of dead-reckoning, there is a prediction model at the sender's side to predict the position of this local player as displayed by the remote player. When the error between the predicted model and the real position exceeds some threshold error value, the player sends an update message to the remote players. Normally this threshold error is fixed for the entire duration of the game and for all objects. In the local-lag approach, we delay the display of local messages for a certain time, hoping that during this lag time the update message from a local player will have reached a remote player and hence consistency will be maintained. This local-lag value is fixed and is the same for all the objects and for the whole duration of the game. A more flexible and adaptive approach is therefore preferred.

4. A HYBRID CLIENT - SERVER APPROACH

From the above discussions, it becomes clear that both server-centric and client-centric approaches have advantages and disadvantages. For this reason, we propose a hybrid approach combining thin and heavy client architectures to benefit from their advantages in the same system. This combination is made adaptive in that the system can decide at runtime according to the game and context requirements what part of the game logic is executed on the client side and what part is executed on the server side. In the case of a high latency and when the game requires strong consistency, a significant part of the game logic has to be executed on the client side in order to apply consistency mechanisms to reach the same state at different terminals. With low network latency, and when the virtual world objects are at a position/speed where small inconsistencies can be tolerated and/or when the capacities of the client's terminal are limited, the client terminal only uses some display mechanism to show the messages directly arriving from the game server.

This hybrid and adaptive approach is particularly suitable for complex games with a variety of objects with different speeds and a network where delays can vary considerably.

A game can be divided into different regions which will have different consistency requirements. In some regions, for example when a player is far away from other players and his/her movement is considerably slow, it is not necessary to have stronger consistency maintenance and only server updates can suffice. On other complex regions where we have many objects near each other with different speed vectors, a stronger consistency maintenance which can only be done on the client side is needed. We call these regions, the critical regions.

We recommend initializing and dynamically changing the values of dead-reckoning threshold and local lag according to three criteria:

1) When the network conditions change. For example, when the communication delay increases because of a sudden heavy traffic in the network or when some jitters occur.

2) When a player enters a critical region [8] in the game's virtual world, we change the values of dead-reckoning threshold and local lag so that we send messages frequently and do not delay messages for a long time to achieve strong consistency in these critical regions.

3) To fix the values of these two parameters according to the requirements of different objects. For example, to have different values for fast moving objects and slow moving objects in the game world.

We now present an adaptive communication protocol to allow client-server mechanisms to change dynamically their behaviour according to the above mentioned criteria.

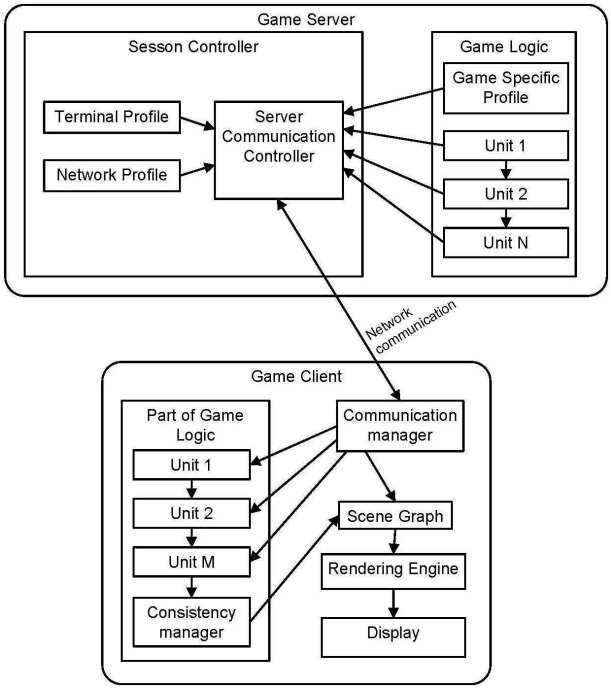

Figure 2. Hybrid architecture for client-server mobile multiplayer games

4.1 Server Side Protocol

As shown in Figure 2, a server has two main components: a Game Logic component and a Communication Controller. The game logic is further divided into different units necessary for the

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8704
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8704

successful execution of the game world. The Server Communication Controller is responsible for the communication with different clients (players) playing this game. While communicating with different clients, it sends the messages to the clients according to three criteria:

1. If a client, to whom the server is sending the message, is having limited memory, processing speed or screen resolution, it sends only scene graphs to be displayed directly by the limited capacity device. In the message, it signals to the client that no game logic processing is required on the client side. This information is stored in the terminal profile which registers the capacities of different clients before the game starts.

2. In case the network latency is very high, or there are jitters on the communication delay with a certain client, the server informs the corresponding client about the high latency/jitters. Because of high latency, inconsistencies can occur at different clients. These clients, with this feedback from the server, have to apply the necessary consistency mechanisms which we explain in the next section. The information about the network conditions is stored in the network profile and the server checks it before sending messages to the clients. The network profile is update consistently according to changes in the network conditions.

3) The consistency requirements of a game depend upon different parameters of a player or object in the virtual world such as his/her position, direction and speed etc. For example, if an object is solitary in the virtual world and its speed is not very high, it may not need a strongly consistent view of others. The information about different game zones, e.g. a circle around a player, is stored in the game profile for each player. If a player has no other player in their critical zone, there is no need for strong consistency on the client side. Hence, in this case, the server should signal the corresponding client that it does not need to apply any consistency maintenance mechanism and that only the scene graph should be updated and be displayed by the rendering engine.

4.2 Client Side Protocol

On the client side, there is a Communication Manager component, a Game Logic component, and a Rendering Engine for displaying the game world. The game logic on the client side is only a subset of the game logic on server side, that is M<N in the diagram. The client side game logic contains only those components which are necessary for consistency maintenance. For example, to use dead-reckoning algorithm for consistency maintenance on the client side in case of a car racing game, it is necessary for the client to have a car track component and some other components to do the necessary predictions on the client's side. The communication manager is the component which receives messages from the server and decides according to the signals/information received whether to do some necessary consistency maintenance or not. As mentioned above, the client needs consistency maintenance on its side according to three criteria: Terminal capacity, network conditions and the player's position and speed in the game world.

If consistency maintenance is required on the client side, the communication manager first sends the message to the game logic which processes the message and applies the necessary consistency maintenance algorithms. After that, the message is used to update the scene graph to be displayed by the rendering engine. If no consistency is required, the communication manager sends the message directly to update the scene graph and render it.

Figure 3. Playing the car game on Nokia N93 mobile phone

4.3 Evaluation

We have done a first evaluation of the hybrid and adaptive approach that we propose using a car racing game we developed for this purpose. The game was developed in J2ME, on the top of GASP platform [18] and can be played on java supported mobiles. We have already tested it on Nokia N93 mobile sets connected to a GASP server via Wi-Fi. The game can also be playing using a 3G network. A picture of the game being played on Nokia N93 is shown in figure 3.

We have evaluated client and server side consistencies in case of high and low latencies. Figure 4 shows the dynamic switching of the game architecture during the game run time from thin-client to heavy-client mode and vice versa.

In Figure 4, at time t0, when the latency is low, the game logic and consistency maintenance algorithms run on the server side as represented by the crossed wheel and the client only displays the messages through its rendering engine. At time t1, when the latency is high, the client also executes some part of the game logic necessary for consistency maintenance, represented by an empty circle, to hide the high latency from the user.

At time t2 the latency is low again and the system switches to thin-client mode. At time t3, although the latency is low, but a player has entered the critical region e.g. player's car is approaching the finishing line, therefore the system switches to heavy-client mode to achieve strong consistency in this critical region.

Figure 5 shows our evaluation for three different scenarios for a car racing game using only two cars. On the Y-axis, we draw the distortion between the actual position of the car on the server and its displayed position on the client's mobile phone. The 'Local' curve shows the actual position of a player on the server on which the game logic component is running. As this 'local' player has a

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8704
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8704

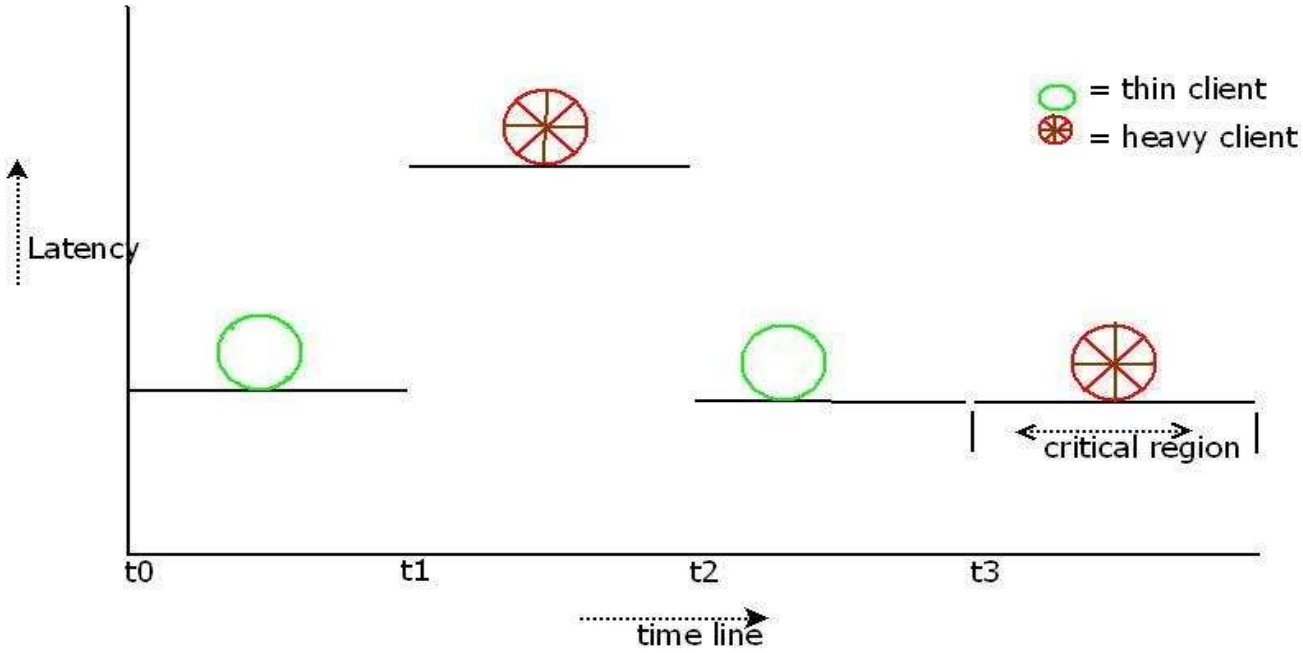

Figure 4. Runtime adaptation of the game architecture

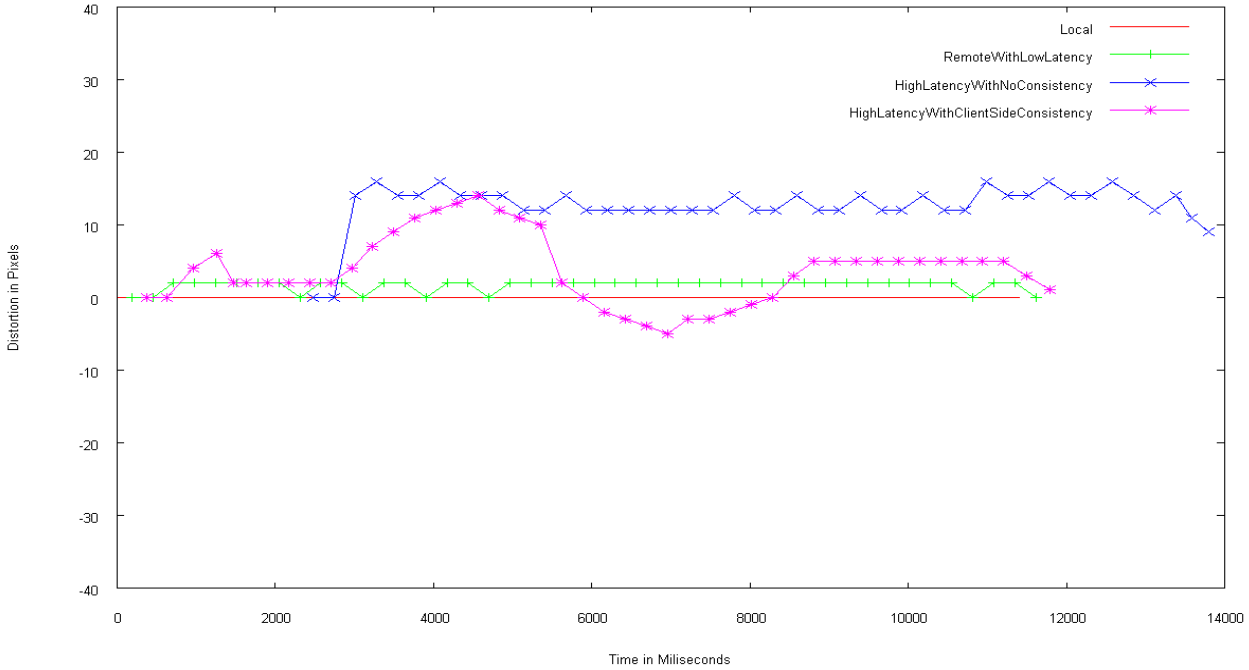

Figure 5. Comparison of state discrepancies in different cases

zero distortion from itself, we draw its positions on value zero on the y-axis and use this value as a reference for the positions of the car displayed on a client's mobile for the other two scenarios.

The "RemoteWithlowLatency" curve shows the distortion of the position of the car on the remote client, when the latency between the client and the server is very low and all the messages coming from the server are shown directly on the client screen, i.e. without applying any client side consistency algorithm and without utilising any game logic on the client side. In this case, the difference between the actual car positions and the displayed car is minimal as can be seen from the figure. This is because the latency between the client and the server is very low and the messages do not arrive very late thereby avoiding the need for applying any consistency mechanism.

In the case of a high network latency (between 1000 and 2000 milliseconds in our implementation), the distortion between the actual car positions and the displayed car on the client side becomes quite visible when no client side mechanism is applied, as shown by the "*HighLatencyWithNoConsistency*" curve. In case of high latency and/or high car speed, we therefore need to apply some client side consistency maintenance to synchronize the data between server and client. In our experimental implementation, we applied our dynamic dead-reckoning and

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8704
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8704

adaptable local-lag approach [8]. This result is shown by the "*highLatencyWithClientSideConsistency*" curve. This curve is *closer* to the '*local*' curve than the one in which the latency is high and no client side consistency maintenance algorithm is used (*HighLatencyWithNoConsistenc*). By *closer*, we mean that the distortion between the actual positions on the server and the displayed positions on the client is low and hence the consistency is improved. This curve is not as straight as the others because when applying prediction algorithms, prediction errors may occur, requiring some recovery time to arrive at the correct position.

From the figure, we can safely argue that in case of very low latencies and when the objects move with a slow pace, messages coming from the server can be displayed directly on the client screen without the fear of high inconsistencies. On the contrary, in case of fast-moving objects and high network latency, we need some mechanism on the client side to apply the necessary consistency maintenance for which some parts of the game logic must reside on the client terminal.

5. CONCLUSION AND FUTURE WORK

In this paper, we proposed a hybrid and adaptive approach to achieve better user experience for multiplayer games on mobile phones. We argued about the need of such an adaptive approach because of the changing wireless network conditions and game's virtual world. We compared the results for both thin-client and heavy-client architectures and showed that a dynamic approach works better than a static approach.

In the future, we intend to evaluate this combined approach on different devices having different capacities. We also would like to test different consistency maintenance algorithms such as critical zone approach using our dynamic hybrid system and using some complex games. We finally intend to implement the communication manager component as part of our Synchronization Medium [9].

6. ACKNOWLEDGEMENTS

This work is a part of the project JEMTU (2006-2008), which aims to design solutions for the technical and psychological issues that slow down the development of multiplayer games on mobile phones (http://proget.int-evry.fr/projects/JEMTU/)

7. REFERENCES

[1] Application protocols. In IEEE Standard for Distributed interactive Simulation. IEEE Standard 1278.1-1995, 1995.

[2] Arsov, I., Preda, M., and Prêteux, F., 2008, MPEG-4 3D graphics for mobile phones, Proceedings First International Workshop on Mobile Multimedia Processing (WMMP'08), Tampa, FL.

[3] Bakhuizen, M. and Horn, U. (2005). Mobile broadcast/multicast in mobile networks.

[4] Chia-chun Hsu, Jim Ling, Qing Li and C.-C. Jay Kuo. 2003. On the design of multiplayer online video game systems. Multimedia Systems and Applications VI ITCom 2003, Proc. of SPIE, volume 5241, Orlando, Florida, USA , September 7-11, 2003.

[5] Cronin, E., Filstrup, B., Kurc, A. R., and Jamin, S. 2002. An efficient synchronization mechanism for mirrored game architectures. In *Proceedings of the 1st Workshop on Network and System Support for Games* (Braunschweig, Germany, April 16 - 17, 2002). NetGames '02.

[6] Eisert, P. and Fechteler, P., 2008,"Low Delay Streaming of Computer Graphics," Proc. International Conference on Image Processing (ICIP), San Diego, USA.

[7] Grimstead, I. J., Avis, N. J., and Walker, D. W. 2005. Visualization across the pond: how a wireless PDA can collaborate with million-polygon datasets via 9,000km of cable. In Proceedings of the Tenth international Conference on 3D Web Technology (Bangor, United Kingdom, March 29 - April 01, 2005). Web3D '05. ACM, New York, NY, 47-56.

[8] Khan, A. M., Chabridon, S., and Beugnard, A. 2008. A dynamic approach to consistency management for mobile multiplayer games. In Proceedings of the 8th international Conference on New Technologies in Distributed Systems (Lyon, France, June 23 - 27, 2008).

[9] Khan, A. M., Chabridon, S., and Beugnard, A., 2007, Synchronization medium: a consistency maintenance component for mobile multiplayer games. In NetGames '07: Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games, pages 99 Melbourne, Australia.

[10] Mauve M, Vogel J, Hilt V, Effelsberg W. Local-lag and timewarp: Providing consistency in replicated continuous interactive media. IEEE Trans. Multimedia, 2004, 6(1): 47--57.

[11] Morán, F., Preda, M., Lafruit, G., Villegas, P. and Berretty, RP., 2007, "3D Game Content Distributed Adaptation in Heterogeneous Environments", EURASIP Journal on Advances in Signal Processing Volume 2007, Issue 2, Pages: 31 – 41.

[12] Nave, I.; David, H.; Shani, A.; Tzruya, Y.; Laikari, A.; Eisert, P.; Fechteler, P., 2008, "Games@large graphics streaming architecture," Consumer Electronics, 2008. ISCE 2008. IEEE International Symposium on , vol., no., pp.1-4, 14-16 April 2008.

[13] Sanna, A. 2007. A Streaming-Based Solution for Remote Visualization of 3D Graphics on Mobile Devices. IEEE Transactions on Visualization and Computer Graphics 13, 2 (Mar. 2007), 247-260.

[14] Zhang L. and Lin Q., "Support Dynamic Network Architecture for Large Scale Collaborative Virtual Environment", accepted for publication in International Journal of Information Technology.

[15] Engel, K., Sommer, O., and Ertl, T. A Framework for Interactive Hardware Accelerated Remote 3D-Visualization. IN PROC. TCVG SYMP. ON VIS. (VISSYM, (2000), 167--177.

[16] Diepstraten, J., Gorke, M., and Ertl, T. Remote line rendering for mobile devices. Computer Graphics International, 2004. Proceedings, (2004), 454-461.

[17] Nurminen, A. m-LOMA - a mobile 3D city map. Proceedings of the eleventh international conference on 3D web technology, ACM (2006), 7-18.

[18] Pellerin R., Delpiano F., Gressier-Soudan E., and Simatic M. GASP: A middleware for multiplayer games in mobile phone networks (in french). In UbiMob '05: Proceedings of the 2nd French-speaking conference on Mobility and Ubiquity computing,; 31 May-03 June 2005.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8704
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8704