Computing and Informatics, Vol. 28, 2009, 619–633

A SITUATION-AWARE CROSS-PLATFORM ARCHITECTURE FOR UBIQUITOUS GAME

JungHyun Han, Dong-Hyun Lee, Hyunwoo Kim, Hoh Peter In

College of Information and Communications
Korea University, Seoul, Korea
e-mail: `hoh_in@korea.ac.kr`

Hee-Seo Chae

Department of Computer Science, USC
LA, USA

Eenjun Hwang

School of Electrical Engineering
Korea University, Seoul, Korea

Young Ik Eom

School of Information and Communication Engineering
Sungkyunkwan University, Suwon, Korea

Manuscript received 25 September 2006; revised 29 October 2007
Communicated by Thomas Fabringer

Abstract. Multi-player online games (MOGs) are popular in these days. However, contemporary MOGs do not really support ubiquity in the sense that a seamless service across heterogeneous hardware platforms is not provided. This paper presents the architecture of the cross-platform online game, which provides a service to users from heterogeneous platforms and is equipped with a situation-aware capability for enabling the users to seamlessly move between heterogeneous platforms. The experimental results through the prototype implementations show the feasibility of the situation-aware cross-platform game.

Keywords: Cross-platform game, situation-aware middleware, conflict resolution

1 INTRODUCTION

Recently, we have seen rapid growth of *multi-player online games* (MOGs) that can accommodate hundreds of thousands simultaneous players. Initially, MOGs were serviced only for PCs. However, they are now running on game consoles such as PlayStation3 and Xbox360, arcade game machines, or even mobile devices.

MOGs have been serviced mostly for a single platform, but *multi-platform* MOGs have recently emerged [1]. In a multi-platform MOG, users from heterogeneous platforms, e.g. a PC user and a cell phone user, can interact. Figure 1 illustrates its ideal environment, where a single game service is provided to multiple platforms, on which their own client game engines are built.

The goal of the research work presented in this paper is to develop a *situation-aware* multi-platform MOG. An evident instance of situation is the game player's *location*, and we propose a smart MOG which can detect changes of the game player's location and then can provide a *seamless move* between heterogeneous platforms. Suppose that, for example, a mobile game player enters home. If the player wants to continue the game on PC which has a better display device, the game should be seamlessly moved to PC with the current game status preserved. Similarly, a game player at PC can seamlessly change the game platform into a mobile device when the player has to leave the PC. When a multi-platform MOG supports such a seamless move between heterogeneous platforms, we name it a *cross-platform game*. This paper presents the architecture of a *situation-aware middleware*, designed and developed for a cross-platform game.

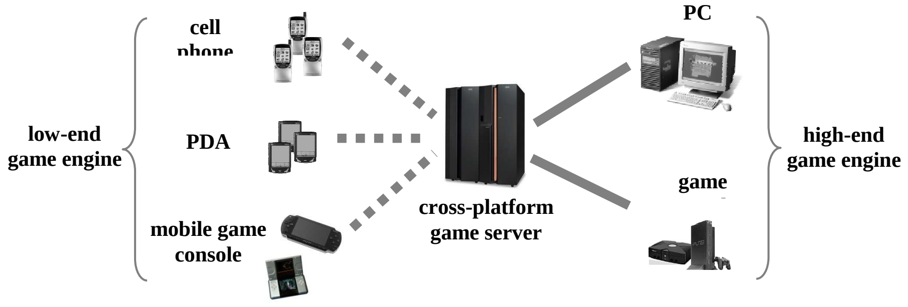


Fig. 1. Multi-platform online game

The cross-platform game is a good instance of *ubiquitous computing*, which is expected to provide a consistent service anytime and anywhere, using any available devices and networks. The cross-platform game presented in this paper can be called a *ubiquitous game*. In designing a ubiquitous game, it is important to note that multiple devices may conflict for a single service. The cross-platform ubiquitous game service must be operated intelligently in order to resolve the conflict between multiple devices, and to provide the best environment to the game player. This paper presents a conflict-resolution model for a ubiquitous game.

The organization of the paper is as follows. Section 2 reviews the related work. Section 3 presents the cross-platform game scenario, and Section 4 discusses the core component of the cross-platform game: situation-aware middleware based on sensor networks. Section 5 discusses the conflict resolution model. Section 6 presents the implementation issues. Finally, Section 7 concludes this paper.

2 RELATED WORK

There have been lots of research works proposed to support mobility in the ubiquitous computing environment. In the domain of personal communications such as mobile telephony, many approaches have been proposed, and a good example is the Berkeley ICEBERG project [2]. In the heterogeneous network environment including cellular networks, pager networks, and IP-based networks, the ICEBERG project aims at 'any-to-any communication' service, which refers to the ability to support communication between different types of devices effectively, and 'personal mobility' support which treats people, rather than devices, as communication endpoints. The latter is achieved by assigning every person a unique ID and mapping between the specific device ID and unique ID through a name mapping service. In the work, however, the cross-platform issue is not properly addressed.

Takasugi et al. [3] proposed a seamless service platform for a ubiquitous network environment. The platform supports 'service migration' which continues a service even if nodes are changed. It is achieved by using a virtual network constructed with seamless proxies that relay data between applications and keep connections among them. Their work focused on mobile nodes acting as servers, but nodes running client programs are not the highlight of the approach.

Cui et al. [4] aimed to support user mobility in the ubiquitous computing environment, and proposed a mechanism to preserve the user's access to the same service despite changes of the accessing host. In a middleware layer, they built mobility functions for handoff management and service instantiation across heterogeneous computing platforms. They tested the system on mobile video and audio players. In contrast, the major application of the work proposed in this paper is in the online games. Further, the proposed middleware has a different architecture from the one of Cui et al. [4] and includes distinct functionalities such as conflict resolution.

The concept of ubiquitous game has recently emerged and explored the merger of ubiquitous computing and computer game. A recent report on the ubiquitous game can be found in [5]. However, previous works have focused on various sensing technologies, for example, for capturing physical interaction, blurring of physical and virtual worlds, etc. Unlike the previous works, this paper presents the first attempt to develop the core components of the sensor network-based situation-aware cross-platform MOG such that an adaptive, seamless service can be provided when a game player moves between heterogeneous platforms.

There have been efforts to apply grid computing techniques to massively multi-player online games (MMOGs). A key element in successful grid collaboration is resource management. Dumitrescu and Foster [6] proposed GRUBER, an architecture and toolkit for resource usage service level agreement (SLA) specification and enforcement in a grid environment, focusing on computing resources such as computers, storage, and networks. On the other hand, there is another line of efforts to develop a new class of applications characterized as real-time online interactive applications, such as online games, based on the grid computing platform [7]. To support the requirements of those applications, advanced services for resource allocation, monitoring and planning are required at the resource management level. Siddiqui et al. [8] introduced a new 3-layered negotiation protocol for advance reservation of the grid resources in order to optimize resource utilization and QoS constraints while generating the contention-free solutions. In the broad sense, the conflict resolution model proposed in this paper may be taken as an extension of the resource management.

3 SITUATION-AWARE CROSS-PLATFORM SERVICE: A UBIQUITOUS GAME SCENARIO

We distinguish between *context* and *situation*. A context represents a state of a device at a specific time while a *situation* is a context augmented with *semantics*. For example, given a device 'at a location,' the context may simply mean its 3D coordinates (x, y, z), but the situation may inform us that the device is in a living room of a house. Ubiquitous game should be able to provide a situation-aware service for a player, using the best platform among the available ones. This is called a *situation-aware cross-platform service*.

This section presents a scenario for situation-aware cross-platform game service, with two game platforms, PDA and PC. Figure 2 illustrates the scenario. The game clients (PDA and PC) and the *game server* are arbitrated and synchronized by the *situation-aware middleware*. The middleware collects raw contextual data through the *sensor network*. Suppose that a PDA game player enters home. Then, the sensor network recognizes the new context (Step 1 in Figure 2) and communicates it to the middleware (Step 2).

The middleware processes the context data, and identifies the situation where PDA and PC are at the same location. If reasoning with the situation data recommends that it is better to continue the game service on PC, the middleware issues an inquiry to PDA (Step 3). The inquiry would be asking the user whether to move

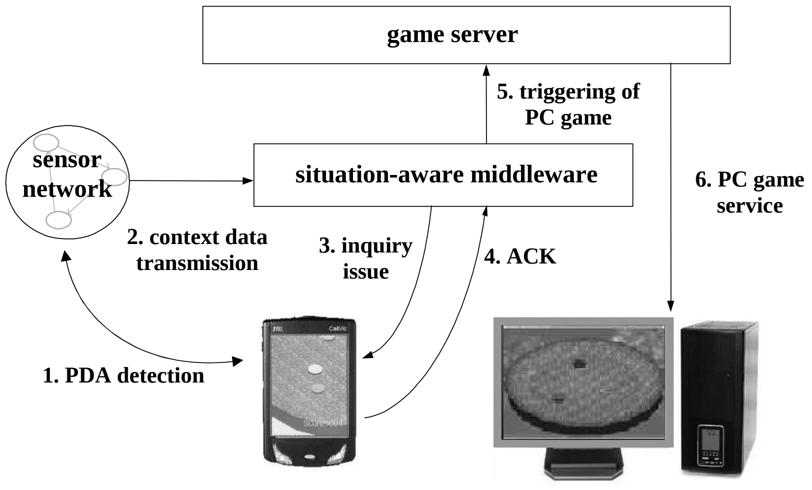


Fig. 2. A cross-platform game scenario

to PC. The user may want to continue the game on PC which has a better display device. When the PDA user agrees to move to PC (Step 4), the middleware triggers the game server (Step 5) in order for the cross-platform game to run at PC (Step 6).

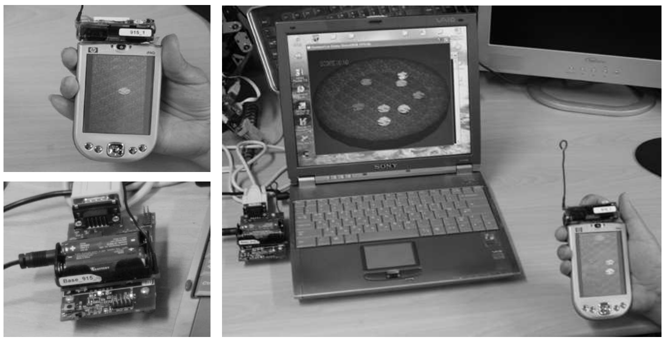


Fig. 3. Cross-platform game implementation with PDA, PC and sensor nodes

Figure 3 shows a prototype cross-platform game (named 'Bumping Pucks') implemented in PDA, PC, and sensors. In the prototype implementation, both of the situation-aware middleware and game server are located in a PC. However, they can be disconnected, and are designed and developed to communicate through TCP/IP. A sensor node (Crossbow's Mote [9]) is attached to PDA, as shown in Figure 3 a). When the PDA game user approaches PC, the gateway sensor (in Figure 3 b)) connected to the PC detects it, and upon the user's acknowledgement, game playing at PDA *seamlessly* moves to PC, as shown in Figure 3 c).

4 SITUATION-AWARE MIDDLEWARE ARCHITECTURE

The situation-aware middleware handles all situation-related issues such as user mobility, and therefore the game server is freed of the burden to be situation-aware. Figure 4 shows the main components of the situation-aware middleware. The mission of the middleware is three-fold and is carried out by *situation registrar*, *situation manager*, and *conflict resolver*, respectively.

The situation registrar processes raw contextual data passed through the sensor network, and then generates and registers situation data into the *situation database*. The situation manager performs reasoning with the situation data, and generates *actions* to change the devices' states. In the process of the situation management, some conflicts may arise among the devices. Then, the conflict resolver is invoked. According to the suggestion from the conflict resolver, the situation manager determines which services are provided in which devices.

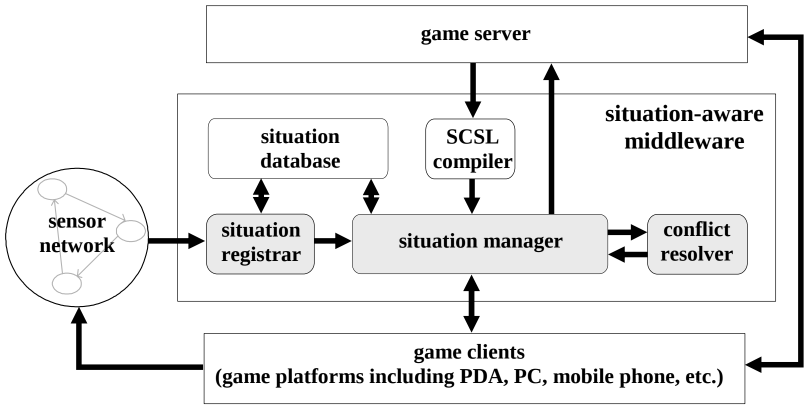


Fig. 4. Situation-aware middleware architecture

A situation is defined per device, and is stored as a tuple `<D, T, S, V>` in the database. A situation `<D, T, S, V>` implies that, at time `T`, device `D`'s state `S` has been set to value `V`. Examples of the states include location of the device, whether the device is playing a game, the URL of the game playing on the device, etc.

Suppose that we have two PCs (PC_1 and PC_2) in the ubiquitous game environment and the situation-aware computing starts at time 0. Then, the situation registrar creates the following situations for PC_1, and stores them in the database.

```
<PC1, 0, location, HOME>          // s1
<PC1, 0, play, OFF>               // s2
```

The above situations say that PC_1 is home and does not run any game. For PC_2, we have the same situations.

```
    <PC2, 0, location, HOME>          // s3
    <PC2, 0, play, OFF>               // s4
```

Suppose that, according to the scenario presented in Section 3, the PDA game player enters home after 5 seconds. On the PDA, the cross-platform online game ‘Bumping Pucks’ is run. Then, the following situations are generated for PDA, and are registered in the database.

```
    <PDA, 5, location, HOME>          // s5
    <PDA, 5, play, ON>                // s6
    <PDA, 5, game, BUMPING-PUCKS>     // s7
```

The above situations say that PDA is home and runs the game ‘Bumping Pucks.’ Upon the new situations in the database, the situation manager is invoked to determine whether the situation requirements are satisfied and, if so, to create appropriate actions. In principle, the situation manager is a *function*, which reasons with the situations in the database.

For a ubiquitous game designer, who is not a well-trained programmer in general, the function may not be easy to code. As an easy authoring tool for the game designer, we have adopted and simplified ‘situation-aware contract specification language (SCSL)’ proposed by the first author of this paper [10, 11]. For moving game play from PDA to PC, the following SCSL code is created by the game designer and stored in the game server.

```
Situation_Aware_Object {
    [activate at (PC.locaton=PDA.location)] move(PC,PDA)
    RequiredResources(CPU(A100,B90,C30),
                      MEMORY(400MB),
                      GPU(A100,B100,C85),
                      // other requirements
                      )
    // other specifications such as QoS
} Move_playing_game ;
```

The above SCSL code simply states that action ‘`move(PC,PDA)`’ is performed when the situation data confirm that PC and PDA are in a location. Further, note that, in order for a PC to play the game, it must meet a set of system requirements, which is specified in ‘`RequiredResources`’. (The system requirement will be discussed in Section 5.) With SCSL, the game designer can specify required actions to be performed when some situation requirements are fulfilled. For more detailed descriptions of SCSL, readers are referred to [10, 11].

When the system starts, the SCSL code is, in advance, fetched to the middleware and compiled into a Java code by the SCSL compiler, as shown in Figure 4. The Java code performs the platform-move by executing a set of pre-defined methods, an example of which is shown in Appendix A: The idle PCs located around the PDA's position are collected and passed to the conflict resolver. When the best-fit PC is suggested by the conflict resolver, the user is inquired whether to move to the PC.

If the user agrees, the cross-platform game gets run at the PC, according to the mechanism shown in Figure 2. As a result, the states of PC and PDA are changed, and the situation manager updates the situation database. Suppose that the game play moves from PDA to PC_2. Then, we have the following situations in the database: s_6 defined for PDA has been updated to declare that the game does not run at PDA any longer; For PC, s_4 is updated and s_8 is newly created.

```
<PDA, 10, play, OFF>                // s6
<PC2, 10, play, ON>                 // s4
<PC2, 10, game, BUMPING-PUCKS>      // s8
```

5 CONFLICT RESOLUTION

In the ubiquitous game environment, a conflict occurs when multiple devices are available for a game service. Suppose that, when a mobile game player enters home, there are multiple PCs that can run the online game, i.e. multiple PCs conflict toward a game service. The conflict resolver shown in Figure 4 handles such a conflict for providing the *best* environments to the game players.

Grade	chip	clock	# of cores
A	Core2 Extreme QX6800	3.0 G	4
B	Core2 Duo E6600	2.4 G	2
C	Pentium D 945	3.4 G	2
D	Pentium 4 2.8	2.8 G	1

Table 1. CPU grades

Grade	chip	vertices per second	shader model
A	eForce 6800 Ultra	600 million	3.0
B	GeForceFX 5950 Ultra	356 million	2.0
C	GeForce4 Ti 4400	125 million	1.0
D	GeForce2 MX400	25 million	-

Table 2. GPU grades

Note that there are typically many system requirements for a game, including CPU and memory specifications, GPU capabilities, etc. For the sake of simplicity, let us consider only CPU and GPU, and classify them into 4 grades. Tables 1 and 2 show the grades of the popular CPUs and GPUs, respectively, with a representative chip for each grade.

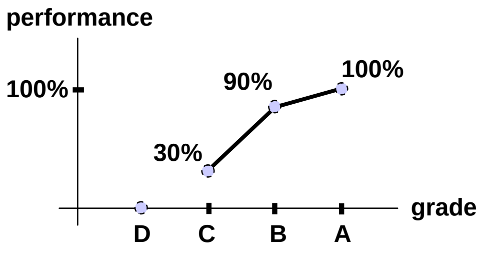


Fig. 5. CPU requirement graph

The game 'Bumping Pucks' on PC performs a good amount of physical simulation. Collision detection and resolution among the pucks are absolutely required, and furthermore, damaged pucks need the effect of mesh fracturing[3]. As the minimum CPU requirement, 'grade C' is specified. The game shows satisfactory and best performances with 'grade B' and 'grade A' CPUs, respectively. These CPU requirements are specified in '`RequiredResources`' of the SCSL code in Section 4, and Figure 5 visualizes them in a graph.[1]

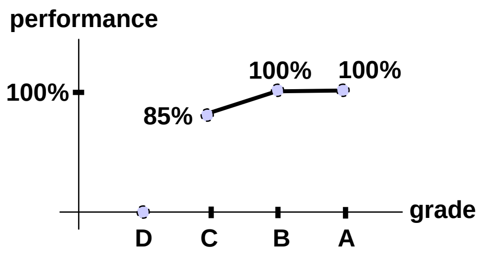


Fig. 6. GPU requirement graph

'Bumping Pucks' is not GPU-intensive. However, its rendering engine is developed using shaders, and 'Bumping Pucks' does not run on 'grade D' GPU, which does not support shaders. Therefore, the minimum GPU requirement is set to 'grade C.' Figure 6 shows the graph of GPU requirements, specified in '`RequiredResources`' of the SCSL code in Section 4.

For a cross-platform game, we need to define a *suitability function* of the hardware requirements. A simplest form of such a function is a linear combination of every hardware component's suitability. Suppose that the function for 'Bumping Pucks' is $2C+G$, where C and G measure the CPU-suitability and GPU-suitability, respectively. The weight associated with the CPU-suitability is 2, and implies that the game is CPU-critical, not GPU-critical.

[1] The PDA version of 'Bumping Pucks' handles collision only, and does not support mesh fracturing, which is a very expensive effect. Instead, a set of textures is used to represent the degrees of damage.

PC	CPU grade	GPU grade	estimated suitability
1	C	A	$2 \times 30 + 100 = 160$
2	A	C	$2 \times 100 + 85 = 285$

Table 3. Conflict resolution model

When a PDA game player enters a room, where two PCs are idle, the PCs compete for serving the game. The CPU and GPU grades of the PCs are shown in Table 3. For each candidate PC, the suitability function is estimated. Let us estimate the suitability of PC_1. The graphs in Figures 5 and 6 show that the CPU-suitability of PC_1 is 30 % and its GPU-suitability is 100 %. According to the suitability function $2C + G$, the overall suitability of PC_1 is then estimated as $2 \times 30 + 100 = 160$. Similarly, the suitability of PC_2 is estimated as 285, as shown in Table 3. Therefore, the conflict resolver recommends that PDA game moves to PC_2, not to PC_1, because PC_2 better fits to 'Bumping Pucks'.

The graphs in Figures 5 and 6 are discrete. Suppose that a hardware component's grade is above the minimum requirement *and* lies in an interval of discrete grades. Then, linear interpolation is used to determine its suitability. When the estimated suitability of every PC falls below some pre-specified threshold, the PDA game does not move to any PC. Of course, when there is no idle PC, no move is triggered at all.

6 IMPLEMENTATIONS

The hardware components of the proposed system include an HP PDA, a desktop PC, and Crossbow's Mote sensors [9]. Microsoft Embedded Visual C++ was used as a development platform for PDA programming. The situation-aware middleware was implemented in Java 2. The situation database is currently implemented in IBM's TSpaces [12]. The time delay for platform move is approximately 2 seconds.

For the proof-of-concept implementation, we have developed a simple TCP/IP-based game server, which follows the traditional architecture. Its modules consist of

1. the *game container*, which provides the game service API and the actual game logic is mounted on,
2. the *gateway server* working as a request-response broker, and
3. the *back-end system* the main part of which is the game database.

We have also developed a high-end game engine for PC and a low-end engine for PDA. The high-end game engine is developed using shaders, especially for implementing the fracturing special effects. The low-end engine is built upon OpenGL ES (OpenGL for Embedded Systems) [13], which is a standard interface for the 3D acceleration chips [14, 15].

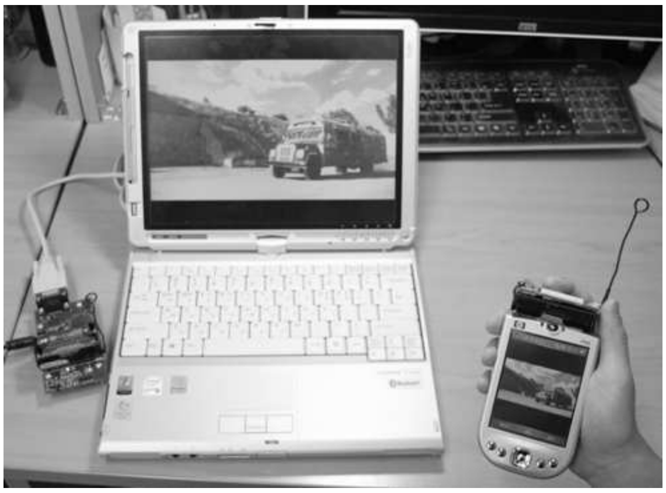


Fig. 7. Cross-platform movie

Even though OpenGL ES is currently adopted by most of mobile 3D chips, it may compete, for example, with Microsoft Direct3D Mobile, a COM-based subset of Direct3D 8, as their desktop ancestors (OpenGL and Direct3D) have done. In order to encompass such various low-level APIs, an abstract/mid-level API (working as a wrapper) is designed, and in fact the low-end engine is built upon the abstract API. Currently, only OpenGL ES implementation in software is connected to the abstract API. However, when it is replaced by a hardware implementation or even by a new low-level API such as Direct3D Mobile, such change will not be visible to the game engine because the abstract API hides all changes in low-level APIs. Mobile game/multimedia consoles include PlayStation Portable (PSP) and Nintendo DS (NDS). Both of them are equipped with 3D acceleration chips. The abstract API of the engine can be layered upon the low-level APIs of PSP and NDS, and therefore the engine can support both PSP and NDS in the future.

The proposed situation-aware middleware can serve not only MOG but also other entertainment applications. A simpler application is *cross-platform movie.* Figure 7 shows such a movie between PC and PDA. When a user playing a movie at PDA approaches a PC, the sensor gateway connected to the PC detects it, and the movie playing at PDA *seamlessly* moves to the PC.

7 CONCLUSIONS

This paper presents research and development results for the situation-aware cross-platform game. Unlike a few existing *turn-based* (not *real-time*) multi-platform MOGs mostly in the casual board game genre, the authors aim at a real-time multi-platform MOG. Furthermore, the game service can be seamlessly moved across heterogeneous platforms. To the best of the authors' knowledge, this work is the first of its kind in the field of multi-platform MOGs.

Note that, in the current work, communications between devices are based on proprietary and somewhat hard-coded protocols. As a future work, more interoperable protocols between ubiquitous devices will be explored. For the purpose, BPEL4WS (Business Process Execution Language for Web Services) [16] and WSDL (Web Service Description Language) [17] are studied to exchange XMLbased messages between devices. We are also considering a more effective conflict resolution method. The current method based on linear combination of hardware components works fine, and is also scalable. However, the conflict resolution is inherently an optimization problem [18], and better techniques from the field can be adopted.

Appendix A

```
typedef struct{
    int CPU;
    int GPU;
}Spec;

public class PDAtoPC extends Thread{
    public void run(){

        int result = 0;
        int bestPC = 0;
        situation PDA;
        PDA = GetSituation();

        result = SearchPCinSDB(PDA.location, idle);

        if (result != 0) {
            int size = sizeof(Spec);
            Spec* idlePCSpec;
            idlePCSpec =
                (Spec*)realloc(NULL, sizeof(Spec)*result);

            for(int i=0; i < result; i++){
                (idlePCSpec+size*i)
                    = getPCinSDB(PDA.location,idle);
            }

            bestPC
                = Conflict_Resolver(idlePCSpec, result);
            if(bestPC != 0)
                Inquire_PDA_to_PC(idlePCSPEC[bestPC-1]);
        }
    }
}
```

Acknowledgement

This research was supported by a Korea University Grant for JungHyun Han, and also was supported by MKE, Korea under ITRC NIPA-2009-(C1090-0902-0046)

REFERENCES

[1] Han, J.—Kang, I.—Hyun, C.—Woo, J.—Eom, Y.: Multi-Platform Online Game Design and Architecture. In Proc. of the 10th IFIP TC13 International Conference on Human-Computer Interaction, Sep. 2005, Rome, Italy, pp. 1116–1119.

[2] The ICEBERG project, `http://iceberg.cs.berkeley.edu/`.

[3] Takasugi, K.—Nakamura, M.—Kubota, M.: Seamless Service Platform for a Ubiquitous Network Environment. NTT Technical Review, Vol. 1, 2003, No. 5, pp. 89–94.

[4] Cui, Y.—Nahrstedt, K.—Xu, D.: Seamless User-Level Handoff in Ubiquitous Multimedia Service Delivery. Multimedia Tools and Applications, Vol. 22, 2004, pp. 137–170.

[5] Bjork, S.—Holopainen, J.—Ljungstr, P.—Mandryk, R.: Personal and Ubiquitous Computing. Special Issue on Ubiquitous Games, Vol. 6, 2002.

[6] Dumitrescu, C.—Foster, I.: GRUBER: A Grid Resource SLA Broker. Lecture Notes in Computer Science, Vol. 3648, 2005, pp. 465–474.

[7] edutain@grid project, `http://www.edutaingrid.eu`.

[8] Siddiqui, M.—Villazon, A.—Fahringer, T.: Grid Capacity Planning with Negotiation-based Advance Reservation for Optimized QoS. Proc. of International Supercomputing Conference, Nov. 2006.

[9] Crossbow Mote,
`http://www.xbow.com/Products/Wireless Sensor Networks.htm`.

[10] Yau, S.—Wang, Y.—Huang, D.—In, H.: Situation-Aware Contract Specification Language for Middleware for Ubiquitous Computing. Proc. of IEEE Workshop on Future Trends of Distributed Computing Systems, May 2003, pp. 93–99.

[11] In, H.—Kim, C.—Yau, S.: Q-MAR: An Adaptive QoS Management Model for Situation-aware Middleware. Proc. of Embedded and Ubiquitous Software Engineering Workshop, Dec. 2004, pp. 972–981.

[12] IBM TSpaces, `http://www.almaden.ibm.com/cs/TSpaces`.

[13] OpenGL ES, `http://www.khronos.org/opengles/`.

[14] ATi IMAGEON series, `http://www.ati.com`.

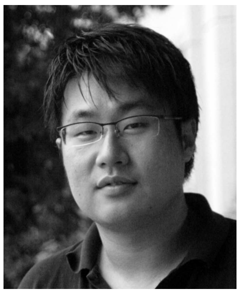
[15] NVIDIA GoForce 3D series, `http://www.nvidia.com`.

[16] Business Process Execution Language for Web Services (BPEL4WS) Version 1.1, 5 May 2003, `http://www.omg.org`.

[17] Web Services Description Language (WSDL) 1.1, W3C Note 15 March 2001, `http://www.w3.org/TR/2001/NOTE-wsdl-20010315`.

[18] Winston, W.: Operations Research. Duxbury Press 2003.

JungHyun Han is a Professor in the College of Information and Communications at Korea University, where he directs the Interactive 3D Media Laboratory and Game Research Center supported by the Korea Ministry of Culture, Sports, and Tourism. Prior to joining Korea University, he worked at the School of Information and Communications Engineering of Sungkyunkwan University in Korea, and at the Manufacturing Systems Integration Division of the US Department of Commerce National Institute of Standards and Technology (NIST). He received a B. Sc. degree in computer engineering at Seoul National University, an M. Sc. degree in computer science at the University of Cincinnati and a Ph. D. degree in computer science at USC. His research interests include real-time simulation and animation for games.

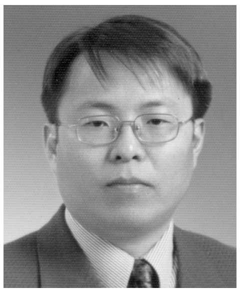


Dong-Hyun Lee is a Ph. D. candidate of the College of Information and Communications at Korea University. His primary research interests are in software engineering on embedded system, value-based software engineering, and ubiquitous computing. He received his M. Sc. degree in computer science from Korea University.

Hee-Seo Chae is an M. Sc. candidate in the Department of Computer Science at University of Southern California. His research interests lie in ubiquitous computing based on wireless sensor networks, multi-agent robotics, and machine learning using probabilistic approaches. He also received his M. Sc. degree in computer science from Korea University.

Hyunwoo Kim is a programmer in Npluto Co. in Korea. He received his B. E. degree in information and communications engineering at Sungkyunkwan University and his M. Sc. degree in computer science and engineering at Korea University. His research interests are in game engine architecture, collision detection and game physics.

Eenjun Hwang received his B. Sc. and M. Sc. degrees in computer engineering from Seoul National University, Seoul, Korea, in 1988 and 1990, respectively; and his Ph. D. degree in computer science from the University of Maryland, College Park, in 1998. From September 1999 to August 2004, he was with the Graduate School of Information and Communication, Ajou University, Suwon, Korea. Currently he is a member of the faculty in the School of Electrical Engineering, Korea University, Seoul, Korea. His current research interests include database, multimedia system, music retrieval, medical image analysis, and web application.



Young Ik Eom received his B. Sc, M. Sc. and Ph. D. degrees from the Department of Computer Science and Statistics of Seoul National University in Korea, in 1983, 1985 and 1991, respectively. He was a visiting scholar in the Department of Information and Computer Science at the University of California, Irvine from Sep. 2000 to Aug. 2001. Since 1993, he has been a Professor at Sungkyunkwan University in Korea. His research interests include system software, distributed computing, mobile agent systems, and system security.

Hoh Peter In is an Associate Professor in the College of Information and Communications at Korea University. His primary research interests are in requirements engineering, value-based software engineering, situation-aware middleware, and software security management. He received his Ph. D. in computer science from the University of Southern California.