International Journal of Computer Games Technology

Networking for Computer Games

Guest Editor: Jouni Smed

Networking for Computer Games

International Journal of Computer Games Technology

Networking for Computer Games

Guest Editor: Jouni Smed

Contents

Hindawi Publishing Corporation
International Journal of Computer Games Technology
Volume 2008, Article ID 928712, 1 page
doi:10.1155/2008/928712

Editorial
Networking for Computer Games

Jouni Smed

Department of Information Technology, University of Turku, 20014 Turku, Finland

Correspondence should be addressed to Jouni Smed, jouni.smed@utu.fi

Received 17 June 2008; Accepted 17 June 2008

Networking has become the main selling point for computer games: commercial games are expected to support multiplaying and the online game sites aim at supporting an ever increasing number of users. At the same time, new game console releases rely heavily on the appeal of online gaming, and a whole new branch of mobile entertainment has emerged with intention to develop distributed multiplayer games for wireless applications. This special issue on "Networking for computer games" focuses on the latest research done on networked computer games and presents five papers exploring different aspects of online multiplayer games.

Multiplayer computer games require both consistent and responsive networking. Consistency is important for maintaining a similar set of data for all players, whereas responsiveness requires that updates to the data are done as promptly as possible. These two requirements, however, are often contradictory and solving this consistency–responsiveness dichotomy lies in the heart of real-time interactive networking. In the first paper "Towards an information model of consistency maintenance in distributed interactive applications," Xin Zhang et al. approach this topic by introducing a framework for analysing the state fidelity of predictive methods.

A massively multiplayer game can have tens of thousands simultaneous players from all of over the world, which means that the scalability of the chosen network architecture becomes critical. Moreover, a massive multiplayer game often requires maintaining a persistent game world, where the game progresses around the clock regardless whether a player takes part in it. The next two papers address this topic. In the second paper "High-level development of multiserver online games," Frank Glinka et al.describe a middleware system called real-time framework, which aims at raising the level of abstraction for the developer of an online game. In the third paper, "ALVIC versus the Internet: redesigning a networked virtual environment architecture," Peter Quax et al. present a generic framework for deploying a massive multiplayer online game using the existing Internet resources.

Mobile gaming and wireless games require a special attention to maintain a continuous and error-free flow information. In the fourth paper "The playing session: enhanced playability for mobile gamers in massive metaverses," Stefano Cacciaguerra and Gabriele D'Angelo introduce a mechanism, based on mimicking the player activities, which is capable of controlling the communication even in a case of a network failure.

The online game sites aim at providing the players more customized content according to the players preference and playing style. This requires methods for analysing the player behavior, which is also important in detecting players who cheat or otherwise misbehave in the game world. In the fifth paper "Visualization of online-game players based on their action behaviors," Keita Iizuka and Ruck Thawonmas present an approach for recognizing different player-type clusters by visualizing the players in-game decisions.

Jouni Smed

Hindawi Publishing Corporation
International Journal of Computer Games Technology
Volume 2008, Article ID 371872, 10 pages
doi:10.1155/2008/371872

Research Article

Towards an Information Model of Consistency Maintenance in Distributed Interactive Applications

Xin Zhang, Tomás E. Ward, and Séamus McLoone

Department of Electronic Engineering, National University of Ireland Maynooth, Maynooth Co. Kildare, Ireland

Correspondence should be addressed to Xin Zhang, xzhang@eeng.nuim.ie

Received 28 January 2008; Accepted 30 April 2008

Recommended by Jouni Smed

A novel framework to model and explore predictive contract mechanisms in distributed interactive applications (DIAs) using information theory is proposed. In our model, the entity state update scheme is modelled as an information generation, encoding, and reconstruction process. Such a perspective facilitates a quantitative measurement of state fidelity loss as a result of the distribution protocol. Results from an experimental study on a first-person shooter game are used to illustrate the utility of this measurement process. We contend that our proposed model is a starting point to reframe and analyse consistency maintenance in DIAs as a problem in distributed interactive media compression.

1. INTRODUCTION

Recent years have seen increasing interest in distributed interactive applications (DIAs). These are applications through which geographically distant end-users exchange information and interact with each other in a shared networked virtual environment. Historically DIAs first saw significant deployment in the military training realm (e.g., Simulation Networking system (SIMNET) [1], Distributed Interactive Simulation (DIS) [2]), and subsequently in the distributed virtual reality community (Naval Postgraduate School Networked (NPSNET) Virtual Environment [3]). However, in the last decade, it is the world of online entertainment systems that has seen the greatest proliferation of such technologies (e.g., Quake [4]). Despite the long evolution of this application class, one of the most persistent problems has been the issue of maintaining a uniform view of the simulation state for all users across the network, that is, the conditions of objects (or entities) and events in the shared environment, under the constraints of limited bandwidth and continuous user interaction.

Consistency refers to maintaining a spatially and temporally identical view of the data across the participating nodes (or hosts) in a DIA [5]. Due to the inevitable network latency which indicates the length of time taken in transmitting a message from one designated node to another [6], perfect consistency in DIAs is impossible to achieve [7] although the effects can be tempered through trading temporal fidelity for state consistency and vice versa [8]. This is known as the "*Consistency-Throughput Tradeoff*," which states that it is impossible for DIAs to have both a consistent and dynamical environment [9]. To deal with the tradeoff, various consistency maintenance mechanisms are employed to ensure a sufficient level of consistency. Generally, techniques used by these mechanisms can be classified into three classes [7]: information management techniques reduce the amount of data that has to be transmitted over the network; time management techniques manipulate time to mask the effect of network latency; system architecture techniques seek to improve the efficiency of processing and disseminating data. In this paper, we focus on one particular group of information management techniques, that is, predictive contract mechanisms that use prediction algorithms to reduce the number of update packets transmitted across the network. These mechanisms have been widely used in military training simulations and computer games [1, 10, 11].

Predictive contract mechanisms maintain controlled inconsistency, or a sufficient level of consistency, by using prediction schemes (e.g., Dead Reckoning [2, 5, 6], Hybrid Strategy Model [12], Nero-Reckoning [13, 14], etc.) to explore information about the future motion of the objects

from contextual dynamics and reduce the frequency of sending entity state updates (ESUs) from the local controlling host to the remote host across the network. The ESUs are only generated and sent out to correct prediction errors larger than a given threshold on some inconsistency metric, while smaller prediction errors are simply ignored. Consequently, the local host only provides an approximated dynamic and encodes it in the ESUs. Predictive contract mechanisms sacrifice the accuracy of the remotely approximated dynamic in return for a reduction in the number of entity state updates and thus save bandwidth and reduce network latency. The performances of such consistency maintenance mechanisms depend mainly on how much the prediction model perceptually matches the real motion of the object on the local host.

Traditionally, performance of a prediction scheme is measured by the frequency of the ESU transmissions required to maintain the inconsistency within the threshold limit. The inconsistency caused by applying the prediction scheme in DIAs has been evaluated and analysed with different metrics of prediction error between states, and their approximations: drift distance (the average absolute error [15]), root mean square error (RMSE), and max norm (the worst error) [16] are all based on spatial difference between states of the same entity on different hosts; phase difference [17] considers temporal difference between the rendering time of the same entity state on different hosts; time-space inconsistency [18] takes into account both spatial distance and its duration. Unfortunately, none of the aforementioned measures gives explicit or quantified analysis of the contribution of the mechanism in helping reduce bandwidth consumption.

In this paper, we introduce a new framework which may aid in this regard and which utilises entropy and mutual information. Mutual information has been used in many other areas to detect and evaluate the dependence between different variables, such as gene expression [19], electrical signals from the brain [20], and so forth. In our model, mutual information is employed to measure the dependence between the real state dynamic and the approximated state dynamic on the local host. The inconsistency induced by discarding prediction errors within the threshold limit is measured as the information loss in the local approximation, which also indicates the theoretical bandwidth saving because mutual information is a direct and quantified measure of the minimal amount of data required to fully describe the interdependence between two variables, namely, the real and approximated motion in the context of DIA. The performance metric of the prediction algorithm is its ability to make use of ESUs to explore information about entity motion and reduce the amount of data required to maintain consistency. By investigating the use of information theory as a measurement of both components, we are able to provide a quantified model to analytically study the "*Consistency-Throughput Tradeoff*" as a problem in lossy source coding on the local host, that is, how good is the local host in providing entity dynamic information and what is the cost for that quality of sharing object motion.

In our model, the complexity of the entity motion is measured by entropy which is calculated from sampled probabilities. There are other advanced and more accurate approaches, such as fuzzy logic and neural networks, used to model object behaviour. An overview of these techniques can be found in [21]. The simple probability model we used here captures the underlying assumption behind all these behaviour models; that users will act similarly under similar circumstances, which makes our approach applicable to any extrapolation method. It is also worth noting that mutual information provides a general measure of the interdependence between the locally generated states for a given entity and those simulated to be rendered remotely. Other statistical measures are also available for measuring the dependence between the real and approximated dynamics. However, most of them only measure specific dependence patterns, such as linearity in the case of the Pearson's correlation. In DIAs, where the motion of the object is usually nonlinear and complicated, such dependence metrics could be misleading [22]. It should be stated that regardless of the measure used, the work reported here is the first such attempt to use any measure of this dependence as a richer measure of compression.

The remainder of this paper is organised as follows. A mathematical background of concepts and methods in information theory that are employed in our model is given in the next section. This is followed by fundamental principles of predictive contract mechanisms and detailed explanations of our information model to formulate local information processing in Section 3. The description of the experimentation is given in Section 4, while Section 5 presents our results and discussion. Finally, the paper ends with conclusions and directions of future work in Section 6.

2. BACKGROUND

We begin with a brief review of the concepts of Shannon entropy and mutual information in information theory and introduce the numerical procedures to estimate them from experimental data. All the definitions and methods here are given in discrete terms, as states of all variables in a virtual environment, however vivid, are finite and discrete.

Consider a random variable X with m possible states $\{x_1, \ldots, x_m\}$, each with probability $p(x_i)$. The entropy $H(X)$ of the variable is defined as [23]

$$H(X) = -\sum_{i=1}^{m} p(x_i) \log p(x_i). \tag{1}$$

Entropy measures the degree of complexity of variable X. In the completely determinant case, some state x^* is such that $p(x^*) = 1$, and all other probabilities are zero, we have $H(X) = 0$. If, on the other hand, there is a universal probability $p(x_i) = 1/m$ for all possible states, the maximal entropy is $H(X) = \log m$. In general, variables with larger entropy are more complex and more unpredictable. From the view of compression, entropy also indicates the minimal length of data required to fully describe the variable.

For two random variables X and Y, the remaining complexity of X given knowledge about Y is defined as conditional entropy:

$$H(X \mid Y) = \sum_{x_i, y_j} p(x_i, y_j)\, p(x_i \mid y_j), \tag{2}$$

where $p(x_i, y_j)$ denotes the joint probability that X is in the state x_i and Y is in the state y_j, and $p(x_i \mid y_j)$ denotes the conditional probability that X is in the state x_i, given Y is in the state y_j. For arbitrary variables, entropy is larger than conditional entropy. The difference between (1) and (2) is the amount of reduced complexity of X from knowing information about Y. Thus, the mutual information $I(X;Y)$ is defined as [23]

$$\begin{aligned} I(X;Y) &= \sum_{x_i, y_j} p(x_i, y_j) \log \frac{p(x_i, y_j)}{p(x_i)\, p(y_j)} \\ &= H(X) - H(X \mid Y). \end{aligned} \tag{3}$$

Mutual information measures the interdependence between the variables X and Y. Notice that entropy is the automutual information between the variable X and itself, that is, $I(X;X) = H(X)$. The mutual information in (3) is also called cross mutual information [20].

All the concepts mentioned here involve knowledge about respective probability functions, which are normally not known in practice. For the purpose of this paper, we use a simple approach to estimate probability and mutual information from experimental data, though other more advanced and complicated algorithms exist [19, 24].

Consider a sequence $x(t)$, $t = 1, 2, \ldots, N$ as a collection of N samples of X at different time instances t. Let r_i be the number of cases that $x(t) = x_i$. The probabilities are estimated as the frequencies of occurrences [19]:

$$\hat{p}(x_i) = \frac{r_i}{N}. \tag{4}$$

Similarly, the joint probability of two variables X and Y can be estimated from two sample sequences $x(t)$ and $y(t)$ with the same length according to

$$\hat{p}(x_i, y_j) = \frac{r_{ij}}{N}, \tag{5}$$

where r_{ij} is the number of cases that $x(t) = x_i$ and $y(t) = y_j$ at the same time. With these probabilities calculated, we can estimate mutual information between X and Y as defined in (3). However, it is known that the estimation of mutual information from limited-length samples is systematically biased due to the finite size effect. The systematic error can be corrected by applying an additional term to the original definition [19, 25]:

$$\begin{aligned} I(X;Y) \approx &\sum_{x_i, y_j} \hat{p}(x_i, y_j) \log \frac{\hat{p}(x_i, y_j)}{\hat{p}(x_i)\, \hat{p}(y_j)} \\ &- \frac{m_{xy} - m_x - m_y + 1}{2N}. \end{aligned} \tag{6}$$

Here, m_x, m_y, and m_{xy} denote the number of different state combinations with nonzero probability. The sample size must be considerably larger than the number of possible state combinations to make a good estimation.

The definitions and estimations presented above allow us to model and analyse predictive contract mechanisms in our new framework, in which maintaining controlled inconsistency is viewed as information sharing with loss. We will use entropy and mutual information to quantify the local information generation and processing in the next section.

3. INFORMATION MODEL

As described before, DIAs use predictive contract mechanisms to reduce data transmission requirements. One of the most common techniques employs a concept called Dead Reckoning [2, 5, 6] to extrapolate state from ESUs. The IEEE DIS standard in particular advocates such methods and further classifies the predictive contract mechanisms into two main components: prediction and convergence [2, 9]. In this article, we only focus on the prediction and reconstruction operated locally. The convergence algorithms as well as network latency, which will certainly affect remote inconsistency, are not included in our current model. Nevertheless, for the convenience of the reader, we will briefly mention the convergence algorithms to give a complete picture of the underlying principles behind predictive contract mechanisms.

The prediction algorithms are the core of the whole mechanism, because they define how the actual entity states are locally packed, with loss, into ESUs, and then reconstructed. In standard Dead Reckoning and its various extensions, multiple-order polynomial functions are used to extrapolate state evolution until the next ESU is generated [2, 26, 27]. More complicated methods involving statistical learning, such as Kalman filters [28] and Neural Networks [13, 14], are employed to improve the performance of prediction. Whether having a closed form formula or not, these algorithms are essentially functions or mappings $f(\cdot)$ from the previous generated ESUs to the anticipated states in the future.

The convergence algorithms define how entity states on remote hosts are corrected, on receiving an ESU, from the inaccurate estimation to its real value, so that the approximated dynamics look more natural and smooth. Currently, polynomial equations are the most commonly used convergence algorithms [9, 26, 29]. Higher-order equations generally generate smoother converging trajectory than low-order equations but they require more computation. The convergence operation is taken after the arrival of an ESU on the remote host to gain better visual perceptual consistency [9] and is thus not considered by the local host in issuing ESUs.

As shown in Figure 1, at each simulation time-step, the local host checks the error between the extrapolated state and the actual entity state. The predicted value is accepted, and the simulation goes on if the error does not exceed the threshold; otherwise an ESU including data required by the prediction algorithm is generated to be sent. Most systems

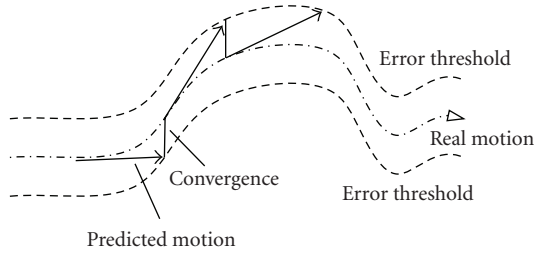


Figure 1: Visual illustration of Dead Reckoning procedures. In this case, we use linear extrapolation and zero-order (or snap) convergence.

send out an update if there is no ESU sent within a timeout period just to inform the remote host that the object is still "alive." These updates are relatively rare compared to the regular ESUs and are not related to local extrapolation, thus they are not considered in this paper. To reconstruct the approximated dynamic remotely, the remote host employs the same prediction model to regenerate the extrapolated states and applies the convergence algorithm on receiving ESUs.

The diagram in Figure 2, with the notation shown, presents our information model in which we seek to reframe the local operations of predictive contract mechanisms as information generation, encoding, and reconstruction processes. The basic idea is that the simulation cycle or "game loop" in a gaming systems context is generating information, that is, it generates updates to entity states at a rate suitable for high fidelity rendering for the local user. By using prediction models and thresholds, predictive contract mechanisms prune the generated information and encode the remainder into ESUs. As such, only an approximated dynamic of the entity is provided by the local host. Less data is required to transmit the approximated dynamic because part of the information is discarded, and the bandwidth saving should equate to the amount of information loss.

3.1. Information generation

As mentioned previously, the shared virtual environment is spatially and temporally discrete. Therefore, measurements of the entity state over time yield the discrete time series $d(k) = \{d(1), d(2), \ldots\}$, where k is the index of simulation time step and the value of $d(k)$ varies within a finite discrete set of entity state values $S = \{s_i\}$. By rendering the entity state at each time step, the local host is generating information about the state. The amount that is dependent on the complexity of the entity's motion can be characterised by a probability function $p_d(s_i)$. The average amount of information generated by the local host at each time step is the amount of uncertainty of the motion, that is, the entropy $H(d)$:

$$H(d) = -\sum_{s_i} p_d(s_i) \log p_d(s_i). \tag{7}$$

In the case of a static environment where the entity state remains at one particular value all along (there is some state s^* such that $p_d(s^*) = 1$), the entropy of this motion would be zero, meaning that there is no information to be shared, and thus no ESU is needed and the remote view of the environment will be consistent with that on the local host. Entity motions with larger entropy require more information in order to fully replicate the state evolution.

3.2. Information encoding

Predictive contract mechanisms reduce the network traffic required for the DIA at the cost of losing state fidelity on the remote host: the local host only provides a pruned dynamic that resembles the real one at some level. This includes two different approximations: ESUs are sent out at a lower frequency than the simulation cycle; each update only contains partial information about the entity motion. Therefore, only part of the information generated on the local host is embedded in ESUs to be sent to the remote host, and the rest is discarded. The tolerable loss of fidelity is controlled by an error threshold. This can be seen as a lossy source coding or lossy media compression.

Let the time series $\hat{d}(k) = \{\hat{d}(1), \hat{d}(2), \ldots\}$ denote the approximated entity dynamic simulated by the local host using predictive contract mechanisms. We use cross mutual information $I(d; \hat{d})$ to measure the amount of information successfully delivered from the real dynamic to the approximated one. The information loss $\mathrm{IL}(d; \hat{d})$ in the process of estimating d as $\hat{d}$ (which is also the reduced bandwidth requirement due to a nonzero threshold) is the remaining uncertainty of the real motion d after we have the approximated $\hat{d}$:

$$\mathrm{IL}(d; \hat{d}) = H(d \mid \hat{d}) = H(d) - I(d; \hat{d}). \tag{8}$$

3.3. Local information reconstruction

To reconstruct the dynamic, the prediction algorithm extracts information about d embedded in the ESUs and interprets it in the form of the approximated state dynamic afterwards. We use time-shift cross mutual information [20] to measure the amount of information utilised by the prediction model. Cross mutual information between any two time series $x(k)$ and $y(k)$ with a time shift τ is defined as [20]

$$\begin{aligned} I(x; y_\tau) &= I(x(k); y(k+\tau)) \\ &= \sum_{x(k), y(k+\tau)} p_{xy}(x(k), y(k+\tau)) \log \frac{p_{xy}(x(k), y(k+\tau))}{p_x(x(k))\, p_y(y(k+\tau))}. \end{aligned} \tag{9}$$

Here, the time shift (or delay) τ refers to the difference between the indices of the time-steps in the two sequences. $I(x; y_\tau)$ is the average amount of information contained in the sequence $x(k)$ that can be learned about $y(k)$ at τ steps later. If the local host is generating an ESU at each time step, we will have the time series $u(k) = \{u(1), u(2), \ldots\}$

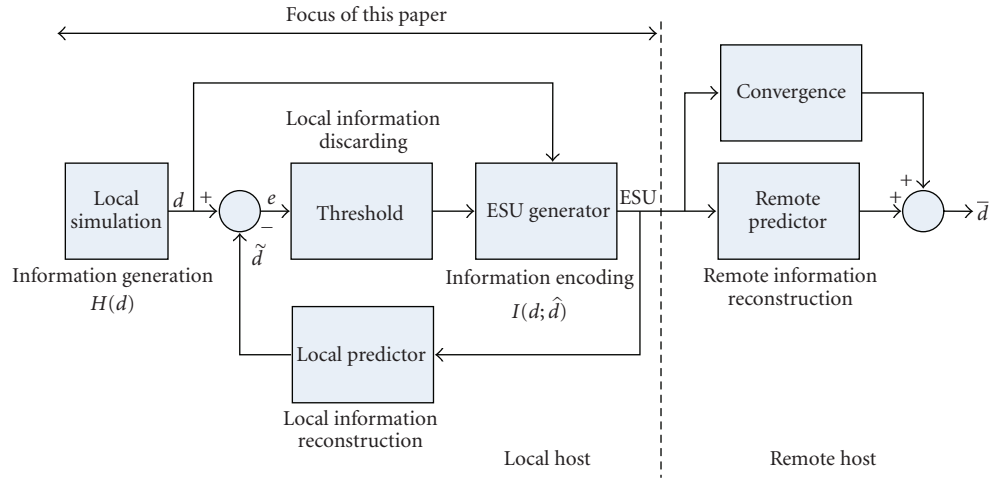


Figure 2: Information model of predictive contract mechanisms. Information generated by local dynamic d is encoded with loss into the ESUs, from which only an approximated dynamic $\hat{d}$ can be reconstructed. The extrapolated dynamic $\tilde{d}$ is compared to the real dynamic to decide whether an ESU is needed to correct the prediction error. $\overline{d}$ is the approximated dynamic reconstructed remotely by the remote predictor and the convergence algorithm.

representing the value of the potential ESU at time-step k. According to (9), the term $I(u;\hat{d}_\tau)$ is the average information that is contained in the ESU sent at time-step k and is used to predict the state at time-step $k+\tau$. Therefore the total information in a single ESU employed by the prediction model should be the sum of all the time-delayed cross mutual information $I(u;\hat{d}_\tau)$, as long as the ESU $u(k)$ is referenced in predicting $\hat{d}(k+\tau)$. For example, in standard Dead Reckoning, only the latest received ESU is used to predict states until the next ESU is generated, so the average information acquired by the prediction model from that ESU is

$$I_U = \sum_{\tau=0}^{l-1} I(u;\hat{d}_\tau), \tag{10}$$

where l is the average time interval between two successive ESUs. l also indicates the number of time-steps that an ESU is employed in prediction, that is, from the time this ESU is generated until the time before the next ESU generation. Thus, (10) calculates the effective information delivered by a single ESU to the local approximation.

With the ESUs being the only source of information to reconstruct entity dynamics, we have

$$I(d;\hat{d}) = \frac{n}{N}\cdot I_U, \tag{11}$$

where n is the number of ESUs during N simulation steps.

Equations (8) and (11) express the core of our information model that information about entity state evolution over time, generated by local hosts, is encoded in ESUs and can be regenerated with some loss by employing prediction algorithms. Actually, (10) and (11) imply an "*Accuracy-Computation tradeoff*" between prediction accuracy against computational and memory resource overhead [9, 26], because computation also takes time and compromises consistency. Simpler models like standard Dead Reckoning only require a single ESU to extrapolate entity states; more complicated methods improve prediction accuracy and further reduce bandwidth consumption, at the cost of additional memory (by referencing longer historical records and more ESUs) and computational resources.

With the model and general procedure described above, we are able to measure how information is generated and processed locally to reduce the amount of data transmitted to maintain consistency, and thus conserve bandwidth. Although the calculations are demonstrated with one-dimensional time series, our approaches can be extended to higher-dimensional movements by using joint probability and joint mutual information [23]. Consequently, calculations in higher-dimensional data would require larger sample sizes to vanish the finite-size effect as (6) states. In the next section, we apply this framework to a multiuser first-person shooter game. The game settings are representative of computer game interactions. The results show that our model is applicable to general predictive contract mechanisms.

4. EXPERIMENTAL DATA

The practical game scenario we use here to show how the proposed framework works is a multiplayer first-person shooter (FPS) game developed using the commercially available Torque Game Engine [26]. The game scenario is shown in Figure 3. The goal of the players in the battlefield is to hold the special "tag" item; in the meantime they can attack each other with their weapons. Players can replenish their health meter in the health-houses. The last survivor holding the "tag" wins. This "deathmatch" scenario is fairly typical in online FPS games. For the convenience of illustration, our numerical study is based only on the x-coordinate of the user. Therefore, the motion of the entity is a one-dimensional dynamic and is recorded as a scalar time sequence $d(k) = \{d(1), d(2), \ldots\}$.

(a) (b)

Figure 3: The FPS game scenario.

The predictive contract mechanism we examine here is Dead Reckoning (DR). We consider both linear and second-order extrapolations. The inconsistency threshold used here is the spatial distance metric. Let $\hat{d}(k) = \{\hat{d}(1), \hat{d}(2), \ldots\}$ be the estimated dynamic. At each simulation time-step k, the linear extrapolation for the current entity state is

$$\tilde{d}_1(k) = d(k_u) + (k - k_u)v, \tag{12}$$

where v is the estimated velocity in the latest ESU generated at time-step k_u. The extrapolated value $\tilde{d}_1(k)$ is accepted if the prediction error does not exceed the given threshold h; otherwise an ESU containing the current state value and velocity estimation is generated based on the real dynamic, that is,

$$u_1(k) = \{d(k), v = d(k) - d(k-1)\}. \tag{13}$$

Here, state $d(k)$ replaces $\tilde{d}_1(k)$ as a correction. For second-order DR, extrapolated entity state and data in an ESU are given by (14) and (15), respectively,

$$\tilde{d}_2(k) = d(k_u) + (k - k_u)v + \frac{1}{2}a(k - k_u)^2, \tag{14}$$

$$u_2(k) = \{d(k), v = d(k) - d(k-1), \quad a = d(k) + d(k-2) - 2d(k-1)\}, \tag{15}$$

where a is the estimated acceleration.

In our experiment, two players are asked to play against each other. Our experiments were conducted for the simulation interval $T_s = 100$ ms and varying error thresholds h. On obtaining both the real and approximated dynamics, we examine the information processing as stated in the previous section. Results and discussion are presented in the next section.

5. RESULTS AND DISCUSSION

Figure 4 shows the real and the estimated dynamics for two different thresholds. As expected, the larger threshold leads to less dynamical approximated entity states, which contains less information about the real dynamic. From state trajectories, it is clear that the two extrapolation methods used here make little difference in approximating entity motion when the threshold is small. The second-order extrapolation causes more intense oscillations when the threshold is large, since more previous states are referenced in order to estimate the acceleration. The effect on prediction errors takes longer time to vanish in "high jerk" motions such as our FPS game.

Figure 5 shows the estimated probability function of the entity states. The probability is unevenly distributed among all possible states. States with significantly higher probability are the places where the player holds its position seeking targets (either the "tag" or the opponent), while those unlikely states are positions the player passes while chasing. The motion of the object generates more information in chasing moments than waiting periods.

Equations (4) and (7) give an entropy $H(d) = 9.2406$ bits. This is the average length of data required per simulation step to fully describe the dynamic. Some of this information generated by the local entity dynamic is lost because errors less than the threshold are simply ignored. The larger the threshold is, the more information is discarded and thus less ESUs are needed to deliver it. In Figure 6, we present information loss and number of ESUs for varying thresholds, along with three different traditional inconsistency metrics:

$$\begin{aligned} \text{drift distance} &= \frac{1}{N}\sum_k |d(k) - \hat{d}(k)|, \\ \text{RMSE} &= \sqrt{\frac{1}{N}\sum_k \left(d(k) - \hat{d}(k)\right)^2}, \\ \text{max norm} &= \max_k |d(k) - \hat{d}(k)|. \end{aligned} \tag{16}$$

In Figure 6, all measurements are normalised, and information loss is presented as a percentage of $H(d)$. It can be seen that for both extrapolation methods, information loss agrees with traditional metrics, following similar trends. The

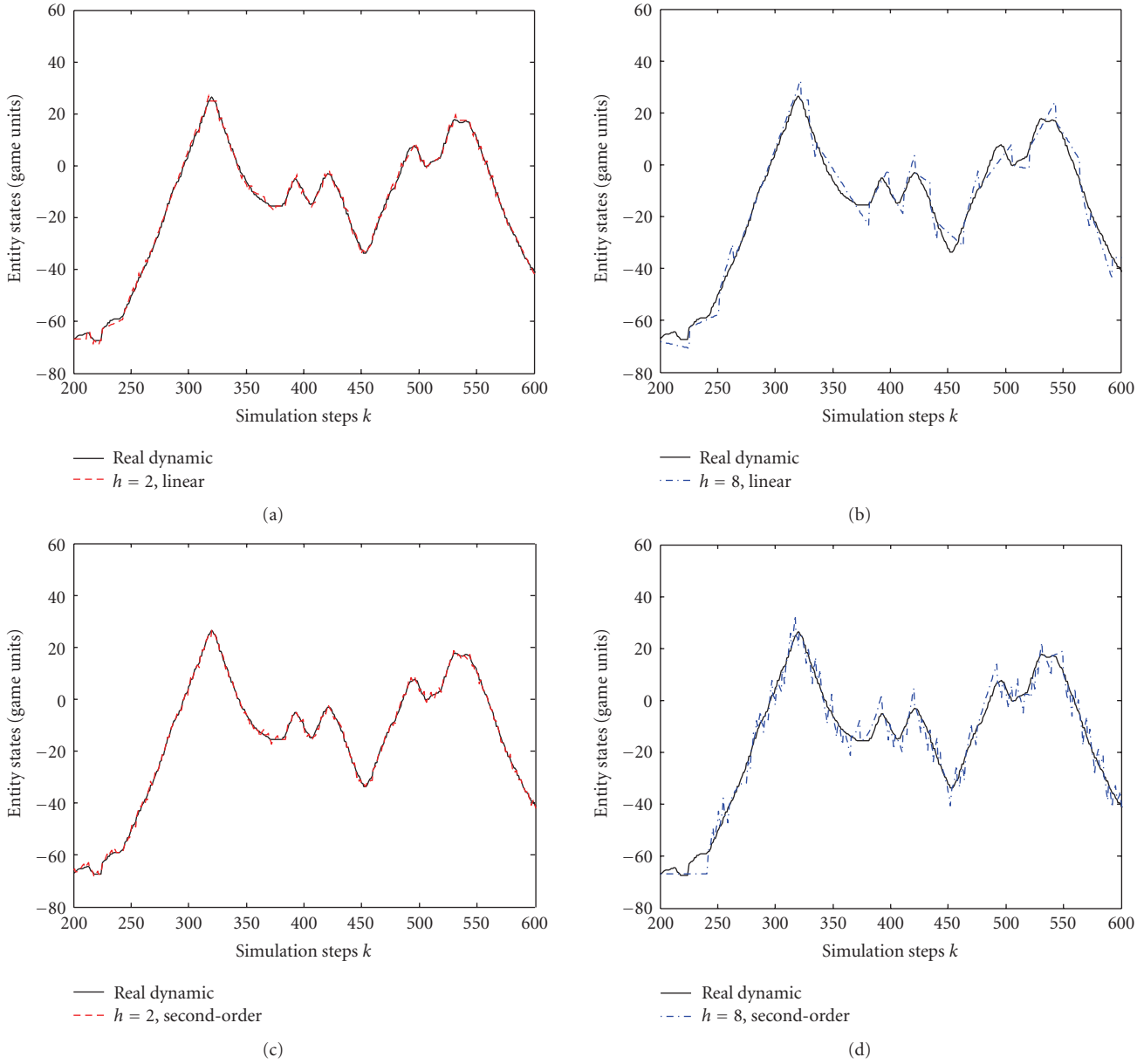


FIGURE 4: The real and estimated dynamics for (a) threshold h at 2 using linear DR extrapolation, (b) threshold h at 8 using linear DR extrapolation, (c) threshold h at 2 using second-order DR extrapolation, and (d) threshold h at 8 using second-order DR extrapolation.

advantage of our information loss measure over the others is that it not only measures to what degree the approximated dynamic resembles the real one but also indicates the bandwidth saved through tolerating inconsistency. In this example, the second-order extrapolation delivers more information to the approximated dynamic than the linear extrapolation.

Figure 6 is also an illustration of the "*Consistency-Throughput Tradeoff*" and provides guidance for a designer to pick a reasonable threshold for a given available bandwidth. For example, with the current prediction algorithms and game scenario, Figure 6 suggests an optimal threshold h between 10 and 20 saves over 80% ESU transmission while losing only around 20% of the information. Larger thresholds lead to little further reduction in ESU transmission, while information loss increases significantly (especially for linear extrapolation); and further reducing information loss would cause significant increase in network traffic.

Another issue worth mentioning is that information loss increases rapidly as threshold increases. As stated before, information loss is the discarded information about entity state evolution and also measures the reduced bandwidth requirement per time-step due to a nonzero threshold. Therefore, we could expect the same reduction rate in the number of ESUs, which indicates the practical bandwidth consumption. However, in Figure 6(a), we can see that frequency of ESU transmission decreases much slower. This observation suggests that ESU transmission can be

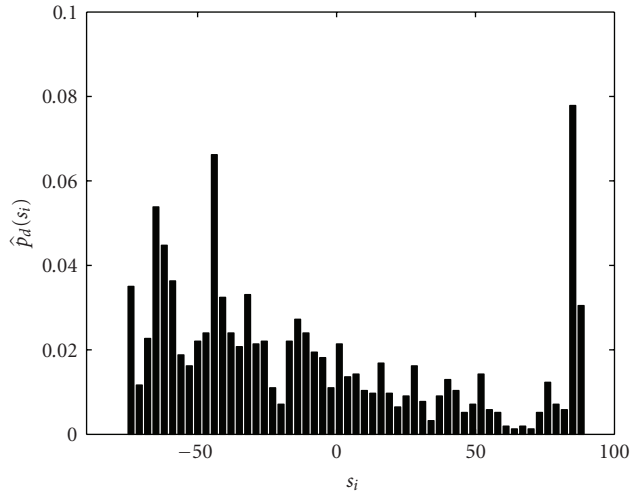
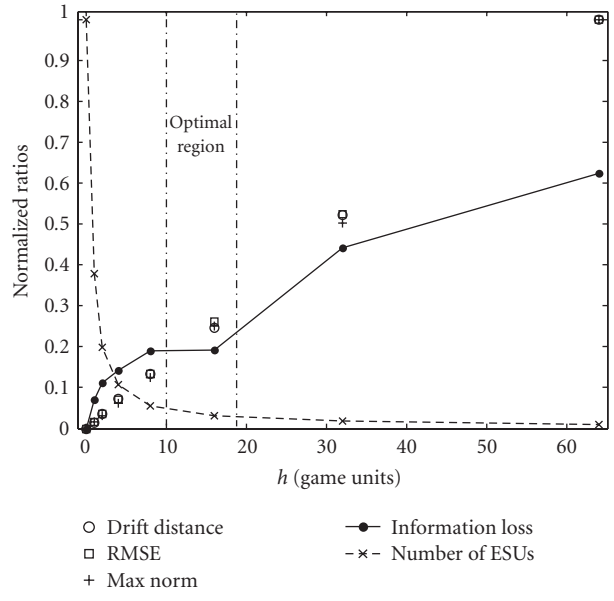
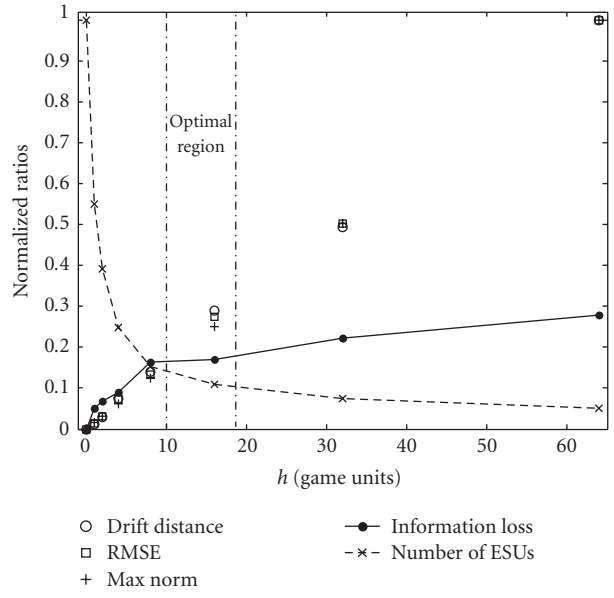


Figure 5: The estimated probability function of the entity states. Possible entity state values s_i vary from −80 to 80.



(a)



(b)

Figure 6: Normalized information loss, three different traditional inconsistency metrics, and number of ESUs for varying thresholds. Results are presented in terms of normalised percentages for (a) linear DR extrapolation and (b) second-order DR extrapolation.

further reduced by applying external compression algorithms: encode them as a signal sequence, and decode them on the remote host. So far, the protocol independent compression algorithm (PICA) has been used to reduce the bit rate in DIAs [30]. It operates by sending only the byte difference between the current ESU packet and a reference packet. But this algorithm does not consider any statistical aspects of the ESUs and is not optimal. The bandwidth used to transmit the ESUs can be further reduced to the theoretical boundary implied by our formulations if statistical compression methods are employed.

We also examine (11) which is how the prediction model acquires information from the ESUs (see Figure 7(a)). Here, the average time-steps l between two ESUs are estimated by

$$l = \frac{N}{n}. \tag{17}$$

Numerical results presented in Figure 7(a) confirm (11) and that information encoded in ESUs is utilised by the prediction model in reconstructing the approximated dynamic. Here, prediction algorithms do not generate or store any information about entity states. Their contribution in saving data transmission is that they interpret information contained in the ESUs, and explore the future entity states from previous motions, and thus reduce redundant update packets. More advanced prediction algorithms match entity dynamics better and can extract more information from the ESUs. This superior performance generally comes from two factors: larger network traffic and more computational resource requirements. For instance, in Figure 7(a), the second-order extrapolation is extracting more information per time step from the latest ESU than the linear extrapolation, at the cost of a larger number of ESU transmissions (see Figure 7(b)) and more memory to restore longer referenced historical states and more computational resources to calculate the extrapolation from (14). Even though computational consumption is becoming more and more insignificant as computers are becoming more powerful, our model still provides a formulation of the ability of the prediction model to utilise ESUs by (10) and an explicit way to deal with the tradeoff between prediction accuracy and computational consumption.

All the formulations in our information model are based on entropy and mutual information between the real and approximated dynamics and are independent of the prediction algorithms applied. Hence, the proposed model is applicable to general predictive contract mechanisms.

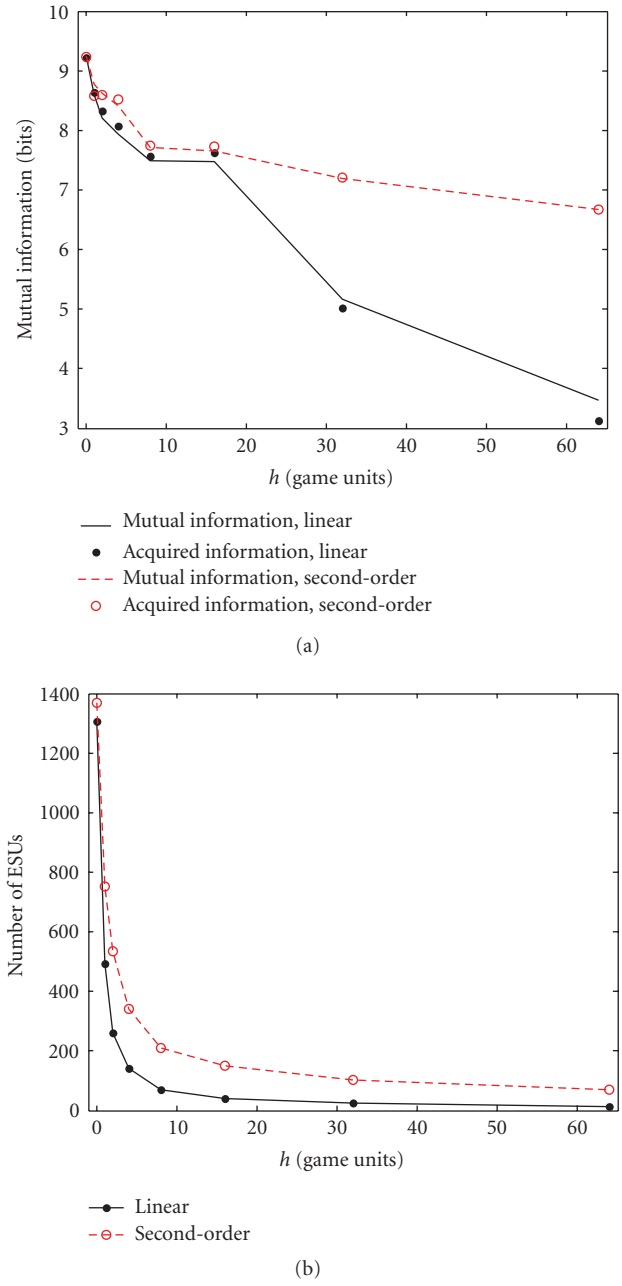


FIGURE 7: (a) Mutual information between the local and remote dynamics, and total information acquired from ESUs in bits per time-step. (b) Number of ESUs for the two extrapolations.

Notice that our information model of predictive contract mechanisms resembles some media compression algorithms. For example, medical images are compressed by encoding quantised prediction errors using differential pulse code modulation (DPCM) [31]. The video compression algorithm MPEG follows similar procedures by employing motion compensation to estimate position of objects in the next frame, and only transmitting the difference transformed by discrete cosine transform (DCT) [32]. Information loss in these media compressions is induced by quantisation. Similarly, in predictive contract mechanisms any prediction error less than threshold is quantised to zero and then discarded. The role of the predictor is to eliminate redundant information between frames in media, or entity states in DIAs, because we can learn some information about future states by exploring previous data. Therefore, it should be pointed out that our framework is a starting point to view consistency maintenance in DIAs as a problem in media compression. Using information theory, we can employ methods in media compression to improve current state-of-the-art communication protocols in DIAs.

6. CONCLUSIONS AND FUTURE WORK

This paper has shown how, by employing information theory, entity state evolution can be viewed as an information generation process, and how predictive contract mechanisms can be modelled as a lossy information compression and reconstruction process. Analytical results show that the local host reduces the amount of data required to maintain some level of consistency by discarding part of the information generated by local entity state evolution. The remaining information is encoded in the ESUs and can be utilised by a prediction model to reconstruct a simpler and less dynamical approximation to the actual entity states.

Through numerical studies, our mutual information metric agrees with traditional inconsistency metrics. Moreover, the advantage of mutual information is that it not only can be seen as an inconsistency metric but also provides the theoretical bandwidth saving which can be achieved by applying prediction models. Our results also suggest that the bit rate in DIAs can be further reduced by applying external compression algorithms that consider the statistical aspects of the ESU sequence. Procedures presented here could shed light on designing optimal prediction algorithms to deal with the "*Consistency-Throughput Tradeoff*" and the "*Accuracy-Computation tradeoff*." Employing information formulations, our model to reframe consistency maintenance as distributed media compression is a novel and promising philosophy in the study of DIAs.

The model as developed in this paper has considered only the information processing on the local host. Within this information-theoretic framework, this is viewed as a form of lossy source compression. The picture of course is far from complete, and in subsequent work we will extend the analysis of predictive contract mechanisms to include the effects of the channel and the decoding or reconstruction of the original state. In particular, we believe that non-ideal attributes of the communication channel such as latency, packet loss and finite bandwidth have insightful interpretation within our framework.

ACKNOWLEDGMENTS

This work is supported by the Irish Research Council for Science, Engineering, and Technology (IRCSET): funded by the National Development Plan. The authors would like to thank Dr. Aaron McCoy for providing the user motion data used in the experimental study.

REFERENCES

[1] J. Calvin, A. Dickens, B. Gaines, P. Metzger, D. Miller, and D. Owen, "The SIMNET virtual world architecture," in *Proceedings of IEEE Virtual Reality International Symposium*, pp. 450–455, Seattle, Wash, USA, September 1993.

[2] IEEE Standard, "IEEE standard for distributed interactive simulation—application protocols," *IEEE Std 1278.1a-1998*, 1998.

[3] M. Capps, D. McGregor, D. Brutzman, and M. Zyda, "NPSNET-V. A new beginning for dynamically extensible virtual environments," *IEEE Computer Graphics and Applications*, vol. 20, no. 5, pp. 12–15, 2000.

[4] D. Kushner, "The wizardry of Id," *IEEE Spectrum*, vol. 39, no. 8, pp. 42–47, 2002.

[5] J. Smed, T. Kaukoranta, and H. Hakonen, "A review on networking and multiplayer computer games," Tech. Rep. 454, Turku Centre for Computer Science, University of Turku, Turku, Finland, 2002.

[6] J. Smed, T. Kaukoranta, and H. Hakonen, "Aspects of networking in multiplayer computer games," *The Electronic Library*, vol. 20, no. 2, pp. 87–97, 2002.

[7] D. Delaney, T. Ward, and S. McLoone, "On consistency and network latency in distributed Interactive applications: a survey—part I," *Presence: Teleoperators and Virtual Environments*, vol. 15, no. 2, pp. 218–234, 2006.

[8] X. Qin, "Delayed consistency model for distributed interactive systems with real-time continuous media," *Journal of Software*, vol. 13, no. 6, pp. 1029–1039, 2002.

[9] S. Singhal and M. Zyda, *Networked Virtual Environments: Design and Implementation*, Addison-Wesley, New York, NY, USA, 1st edition, 1999.

[10] Y. Yu, Z. Li, L. Shi, Y.-C. Chen, and H. Xu, "Network-aware state update for large scale mobile games," in *Proceedings of the 16th International Conference on Computer Communications and Networks (ICCCN '07)*, pp. 563–568, Honolulu, Hawaii, USA, August 2007.

[11] W. R. Johnson, T. W. Mastaglio, and P. D. Peterson, "The close combat tactical trainer program," in *Proceedings of the 25th Winter Simulation Conference (WSC '93)*, pp. 1021–1029, Los Angeles, Calif, USA, December 1993.

[12] J. D. Delaney, *Latency reduction in distributed interactive applications using hybrid strategy-based models*, Ph.D. thesis, Department of Electronic Engineering, National University of Ireland, Maynooth, Maynooth, Ireland, 2004.

[13] A. McCoy, T. Ward, S. McLoone, and D. Delaney, "Multistep-ahead neural-network predictors for network traffic reduction in distributed interactive applications," *ACM Transactions on Modeling and Computer Simulation*, vol. 17, no. 4, article 16, pp. 1–30, 2007.

[14] A. McCoy, T. Ward, S. McLoone, and D. Delaney, "Using neural-networks to reduce entity state updates in distributed interactive applications," in *Proceedings of the 16th IEEE Signal Processing Society Workshop on Machine Learning for Signal Processing (MLSP '06)*, pp. 295–300, Maynooth, Ireland, September 2007.

[15] C. Diot and L. Gautier, "Distributed architecture for multiplayer interactive applications on the Internet," *IEEE Network*, vol. 13, no. 4, pp. 6–15, 1999.

[16] J. J. LaViola, "A testbed for studying and choosing predictive tracking algorithms in virtual environments," in *Proceedings of the Workshop on Virtual Environments*, pp. 189–198, Zurich, Switzerland, May 2003.

[17] J. C. S. Lui, "Constructing communication subgraphs and deriving an optimal synchronization interval for distributed virtual environment systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 13, no. 5, pp. 778–792, 2001.

[18] S. Zhou, W. Cai, B.-S. Lee, and S. J. Turner, "Time-space consistency in large-scale distributed virtual environments," *ACM Transactions on Modeling and Computer Simulation*, vol. 14, no. 1, pp. 31–47, 2004.

[19] R. Steuer, J. Kurths, C. O. Daub, J. Weise, and J. Selbig, "The mutual information: detecting and evaluating dependencies between variables," *Bioinformatics*, vol. 18, supplement 2, pp. S231–S240, 2002.

[20] J. Jeong, J. C. Gore, and B. S. Peterson, "Mutual information analysis of the EEG in patients with Alzheimer's disease," *Clinical Neurophysiology*, vol. 112, no. 5, pp. 827–835, 2001.

[21] E. Frias-Martinez, G. Magoulas, S. Chen, and R. Macredie, "Modeling human behavior in user-adaptive systems: recent advances using soft computing techniques," *Expert Systems with Applications*, vol. 29, no. 2, pp. 320–329, 2005.

[22] W. Li, "Mutual information functions versus correlation functions," *Journal of Statistical Physics*, vol. 60, no. 5-6, pp. 823–837, 1990.

[23] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, John Wiley & Sons, New York, NY, USA, 2nd edition, 2006.

[24] L. Paninski, "Estimation of entropy and mutual information," *Neural Computation*, vol. 15, no. 6, pp. 1191–1253, 2003.

[25] M. S. Roulston, "Estimating the errors on measured entropy and mutual information," *Physica D*, vol. 125, no. 3-4, pp. 285–294, 1999.

[26] A. B. McCoy, *Data-driven modelling approaches for network traffic reduction in distributed interactive applications*, Ph.D. thesis, Department of Electronic Engineering, National University of Ireland, Maynooth, Maynooth, Ireland, 2007.

[27] B.-S. Lee, W. Cai, S. J. Turner, and L. Chen, "Adaptive dead reckoning algorithms for distributed interactive simulation," *International Journal of Simulation Systems, Science & Technology*, vol. 1, pp. 21–34, 2000.

[28] D. Delaney and T. Ward, "A Java tool for exploring state estimation using the Kalman filter," in *Proceedings of IEEE Irish Signals and Systems Conference (ISSC '04)*, pp. 679–684, Belfast, Northern Ireland, June-July 2004.

[29] T. P. Duncan and D. Gračanin, "Pre-reckoning algorithm for distributed virtual environments," in *Proceedings of the 35th Winter Simulation Conference (WSC '03)*, vol. 2, pp. 1086–1093, New Orleans, La, USA, December 2003.

[30] M. A. Bassiouni, M.-H. Chiu, M. Loper, M. Garnsey, and J. Williams, "Performance and reliability analysis of relevance filtering for scalable distributed interactive simulation," *ACM Transactions on Modeling and Computer Simulation*, vol. 7, no. 3, pp. 293–331, 1997.

[31] K. Chen and T. V. Ramabadran, "Near-lossless compression of medical images through entropy-coded DPCM," *IEEE Transactions on Medical Imaging*, vol. 13, no. 3, pp. 538–548, 1994.

[32] International Standard, ISO/IEC 11172, "Information technology—Generic Coding of Moving Pictures and Associated Audio: Systems," 1994.

Hindawi Publishing Corporation
International Journal of Computer Games Technology
Volume 2008, Article ID 327387, 16 pages
doi:10.1155/2008/327387

Research Article

High-Level Development of Multiserver Online Games

Frank Glinka, Alexander Ploss, Sergei Gorlatch, and Jens Müller-Iden

Department of Mathematics and Computer Science, University of Münster, 48149 Münster, Germany

Correspondence should be addressed to Frank Glinka, glinkaf@uni-muenster.de

Received 2 February 2008; Accepted 17 April 2008

Recommended by Jouni Smed

Multiplayer online games with support for high user numbers must provide mechanisms to support an increasing amount of players by using additional resources. This paper provides a comprehensive analysis of the practically proven multiserver distribution mechanisms, zoning, instancing, and replication, and the tasks for the game developer implied by them. We propose a novel, high-level development approach which integrates the three distribution mechanisms seamlessly in today's online games. As a possible base for this high-level approach, we describe the real-time framework (RTF) middleware system which liberates the developer from low-level tasks and allows him to stay at high level of design abstraction. We explain how RTF supports the implementation of single-server online games and how RTF allows to incorporate the three multiserver distribution mechanisms during the development process. Finally, we describe briefly how RTF provides manageability and maintenance functionality for online games in a grid context with dynamic resource allocation scenarios.

1. INTRODUCTION

The development of large-scale, *massively multiplayer online games* (MMOGs) is significantly more complex as compared to the development of casual online games for small user numbers. Since a single server is not capable of serving the high user numbers and the related game processing of contemporary MMOGs, a multiserver design becomes mandatory. This leads to a number of new, complicated design issues, including efficient multiserver communication, object migration between servers, distributed client connection handling, synchronization of data across servers, load balancing, latency issues, and others. A proper addressing of these aspects within the game development process requires high in-house expertise and employs low-level programming and network tools, which makes it time-consuming, risky, and often expensive.

This paper first analyses the basic structure of today's online games and develops a taxonomy of the currently employed development approaches. Based on this analysis, we then describe a novel, high-level development approach that aims at simplifying the development process and improving its productivity while maintaining a high level of flexibility for the developer. Our approach suits a wide spectrum of online games: from traditional single-server games to multiserver MMOGs, with the possibility to enhance a single-server design to a multiserver, multiplayer game. We describe in the detail *Real-Time Framework* (RTF)—a middleware system that supports the proposed high-level approach to game development. RTF, whose first architecture approach and offered development methodology has been introduced in [1, 2], provides integrated solutions for a variety of development and run-time problems. These solutions include the basic communication for a variety of development and run-time problems, ranging from the basic communication functionality to single-server online games to sophisticated solutions for the distribution management of MMOGs. This includes in particular the object-oriented design and efficient transmission of game data structures, interfaces and services that allow the developer to efficiently process an MMOG on multiple servers, integrated monitoring and controlling functionality for games, and management possibilities for the multiserver distribution of MMOGs.

Although there has been previous work targeting most of these problems individually [3, 4], there is a high demand of integrated solutions available as high-level libraries and middleware systems for broad classes of online games. Our RTF-based approach addresses a large variety of online game types, ranging from fast-paced and small action

games to large-scale MMOGs. Moreover, our development approach employs modern Grid computing technologies [5] to facilitate the dynamic usage of system resources, accordingly to changing user demands. This paper presents a comprehensive overview of RTF, motivating and describing its high-level development approach. We present in detail the different parallel processing models offered by RTF for scaling MMOGs, and describe use cases for RTF and how it can be used in a grid environment for on-demand provision of resources. Furthermore, we introduce several new application demonstrators built on top of RTF and report the results of first performance and scalability tests.

The contributions and the structure of the paper are as follows. We describe the basic design of online games that are based on a *real-time loop* in Section 2 and provide a comprehensive analysis of the current game development approaches, with respect to their complexity and flexibility in Section 3. We outline our high-level game development approach, describe the concepts of the supporting RTF middleware, and give an overview over RTF as a development tool in Sections 4 and 5. We show how RTF is employed for multiserver game processing and give an extensive overview of practically proven multiserver distribution mechanisms and the tasks implied for their management in Section 6. We demonstrate how RTF allows to realize a seamless virtual environment and the seamless transfer of game parts between resources in Section 7 and explain how RTF could be used in a grid context with dynamic resource management in Section 8. The paper is concluded by a detailed report on several prototype applications including experimental scalability tests in Section 9 and by summarizing our contributions in the context of related work in Section 10.

2. BASIC DESIGN OF ONLINE GAMES

The majority of today's online games typically simulate a spatial virtual world which is conceptually separated into a static part and a dynamic part. The static part covers, for example, environmental properties like the landscape, buildings, and other nonchangeable objects. Since the static part is preknown, no information exchange about it is required between servers and players. The dynamic part covers objects like avatars, *nonplaying characters* (NPCs) controlled by the computer, items that can be collected by players or, generally, objects that can change their state. These objects are called entities and the sum of all entities is the dynamic part of the game world. Both parts, together, build the game state which represents the game world at a certain point of time.

For the creation of a continuously progressing game, the game state is repeatedly updated in real time in an endless loop, called *real-time loop*. Figure 1 shows one iteration of the server real-time loops for multiplayer games based on the client-server architecture. The figure shows one server, but in a multiserver scenario this may be a group of server processes distributed among several machines. A loop iteration consists of three major steps. At first the clients process the users input and transmit them to the server (step 1 in the figure). The server then calculates a new game state

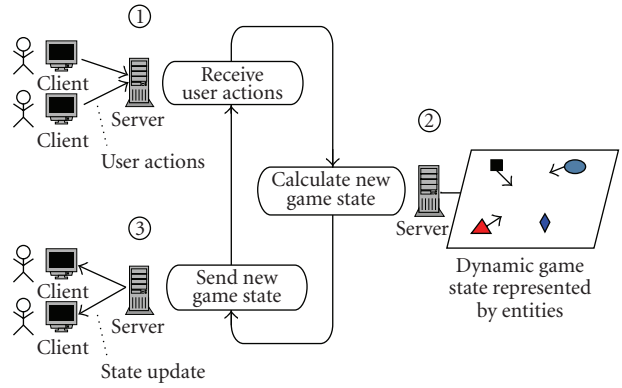


Figure 1: One iteration of the server real-time loop.

by applying the received user actions and the game logic, including the *artificial intelligence* (AI) of NPCs and the environmental simulation, to the current game state (step 2). As the result of this calculation, the states of several dynamic entities have changed. The final step 3 transfers the new game state back to the clients.

When realizing the real-time loop in a particular game, the developer has to deal with several tasks. In steps 1 and 3, the developer has to transfer the data structures realizing user actions and entities over the network. If the server is distributed among multiple machines, then step 2 also implies the distribution of the game state and computations for its update. This brings up the task of selecting and implementing appropriate distribution concepts.

3. GAME DEVELOPMENT APPROACHES

The central part of a game software system consists of the *game state* and the continuous *processing* of the game state. In this paper, we focus on the development of the game state and its processing, rather than on the game user interface, that is, the representation of the virtual environment the player interacts with.

In order to compare different development approaches of the overall distributed architecture of online games, we identify the following three major aspects of online game software systems:

(i) game logic: *entities*, *events* (data structures), and processing rules describing the virtual environment;

(ii) game engine: *real-time loop* which continuously processes (user) events, according to the rules of the game logic, to compute a new game state;

(iii) game distribution: logical partitioning of the game world among multiple servers, computation distribution management according to actual game state, and communication.

The third aspect, game distribution, can be further split up into two levels of distribution: (a) distribution of the user interface and game state processing between client and server, and (b) distribution of game state processing in the multiserver architecture.

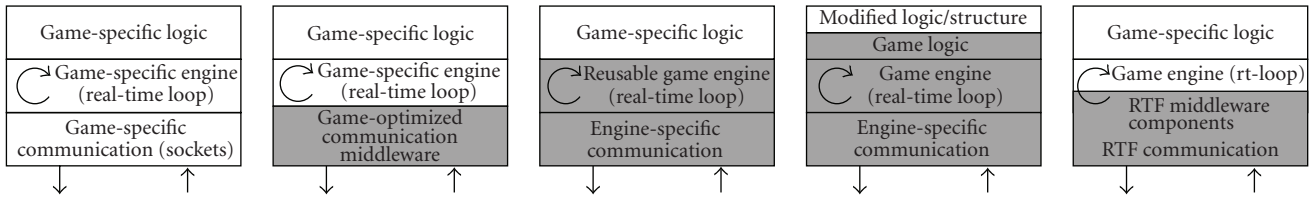


Figure 2: Main approaches to game development. White: self-developed; Gray: using existing components.

These three aspects are treated differently, depending on the requirements and properties of a particular game genre. For example, fast-paced action games rely on efficient communication and engine implementation while using only relatively simple game logic and content. The complexity of the game distribution aspect usually increases with the number and density of the participating users within a game and is thus particularly challenging for MMOGs.

Our classification in Figure 2 distinguishes common approaches (a)–(d) to game development, according to how they treat these three aspects. In each approach, the aspects shown in white are managed by the human developer, whereas the shaded areas are provided automatically by the development system.

(a) Custom development

The most direct approach used for game development is to design and implement the entire software system individually. The development team designs and implements all three major aspects of the game software system: game logic, game engine, and game distribution. This allows the developers to have full control over their code and optimized implementation with focus on the individual performance needs of the game. While the custom development of an entire game is very complex, hence cost-intensive and error-prone, it is sometimes the only way to achieve the particular objectives of the game design because of its flexibility.

(b) Game communication middleware

This approach uses special communication libraries and middleware systems (like Quazal Net-Z [6]) for game development. As shown in Figure 2(b), the game developer employs the middleware to realize the communication between clients and servers in a distributed game while implementing the game engine and logic on his own. Using this approach, the developer has enough flexibility to design and implement the aspects of game logic and game engine while the middleware deals with the game distribution. However, available libraries usually focus on a particular architecture setup, decreasing flexibility of the engine development. Furthermore, this approach has been used only rarely for the development of multiserver-based MMOGs since a pure communication library is not sufficient for these games. A middleware for MMOGs also has to deal with the difficult task of distributing the game processing among multiple servers, for which only a few middleware systems are available (e.g., Emergent Server Engine [7] or BigWorld [8]).

(c) Using existing engine

With this approach, shown in Figure 2(c), an existing game engine, that is, the processing component of a game, is reused to develop a completely new game. This reduces the complexity of development. Some game studios design their game engines primarily for the purpose of reselling and licensing the engine afterwards. Examples of popular and often used engines are the *Quake 3* engine or the *Unreal* engine. However, a particular engine is quite inflexible because it is usually closely tied to a specific game genre.

(d) Game modding

Figure 2(d) outlines the approach of game modding (community jargon for modifying an existing game) via a dedicated interface for programming the game logic. This was first done by hobby developers who modified the actual game content. Nowadays, the creation of mods is based on high-level tools created and also used by the game development studios themselves. Such tools allow the creation of game content by designers with minimal programming effort. The primary aspect of modding is the creation of new game content within the constraints of an existing game logic; hence it is rather inflexible. Nevertheless, modding allows to develop innovative game concepts, and sometimes a mod becomes even more popular than the original game as, for example, the mod *counter-strike* based on Valve's game *Half-Life*.

(e) RTF multiserver middleware

Our real-time framework, as illustrated in Figure 2(e), allows a novel game development approach which provides more processing support than using only a communication middleware, but does not constitute a complete game engine, allowing higher flexibility. Thus, RTF can be classified in between the approaches (b) and (c). The characteristics and usage of RTF, justifying this classification, are discussed in the next sections.

Figure 3 illustrates our taxonomy of the five discussed development approaches with respect to their flexibility and complexity. The most simplicity in terms of distributed

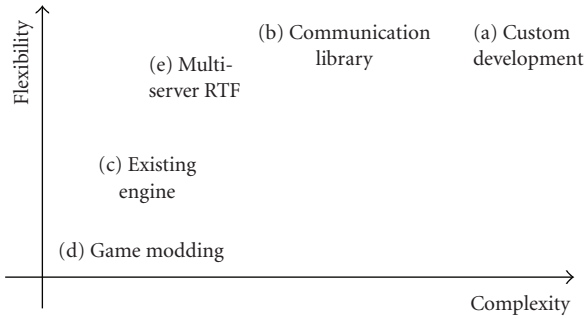


Figure 3: Taxonomy of game development approaches.

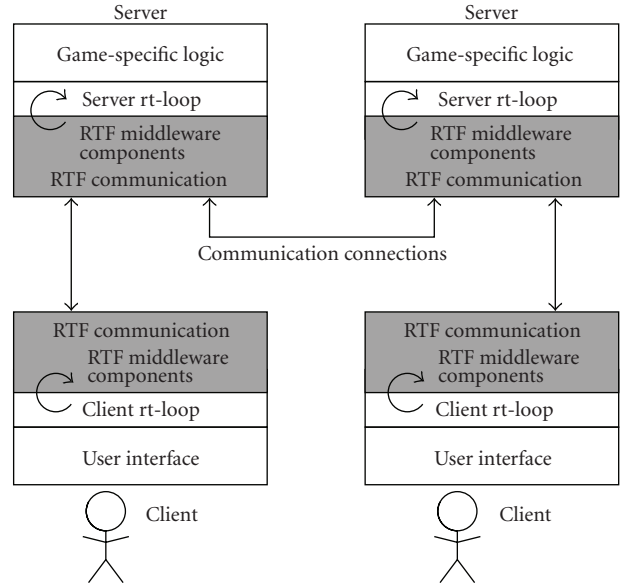


Figure 4: RTF multiserver middleware.

software infrastructure is offered by existing game engines (c) or modding toolkits (d). However, these approaches have the remarkable drawback of being quite inflexible. Obviously, the fully custom development (a) offers most flexibility while being rather complex. The use of special middleware (b) is a promising alternative for particular tasks: its use reduces the complexity of game development. Pure communication support is not enough for MMOGs: for such large distributed systems, the multiserver management is quite extensive and increases the development complexity. As indicated in the taxonomy, RTF is designed to provide the developer with the highest possible flexibility in game design while freeing him from complex low-level implementation tasks in the game development process.

4. RTF OVERVIEW

The real-time framework provides a high-level communication and computation middleware for single-server and multiserver online games. RTF supports both the server-side and client-side processings of an online game with a dedicated set of services which allows developers to implement their optimized engine at a high, entity-based level of abstraction in a flexible manner. Figure 4 shows a generic multiserver example of a game developed on top of RTF.

The RTF middleware deals with entity and event handling in the real-time client loop and the continuous game state processing in the real-time server loop, and the distribution of the game state processing across multiple real-time server loops. The developer implements the game-specific real-time loop on client and server, as well as the game logic, using the RTF middleware to exchange information between the processes.

RTF is based on a modular approach and provides additional components for other aspects of distributed games besides the game processing aspect. Figure 5 shows the components that exist beside the *communication and computation parallelization (CCP) module* which handles the game processing part. The shown components include a module for the persistent storage of game-related information, which is especially relevant for MMORPGs. The *persistency module* allows storing and retrieving entities specified by the application developer in and from a relational database. An *audio streaming module* supports Voice over IP (VoIP) audio communication over RTP (real-time protocol) and provides an interface for setting up channels, switching users between them and to unmute users. Other important aspects of MMOGs, which typically have long-running game sessions, are monitoring and controlling possibilities. The *controlling and monitoring module* is the middleware-endpoint for application developers to receive commands for steering the application at runtime and to report internals about the application status. The developer can define profiles which reflect game-specific controlling and monitoring characteristics on top of this module. There also exist predefined profiles for typical monitoring metrics and controlling tasks in games, and RTF supports some of these predefined modules, for example, reports RTF-internal values like communication characteristics (bandwidth consumption, latency, packet rate) and distribution characteristics (number of clients, number of entities, number of exchanged events), such that the application developer is not required to report such information explicitly.

A modular approach allows RTF to be extensible without a major impact on the existing parts of the RTF middleware. Furthermore, the developer only gets in touch with the modules he wants to use. The remainder focuses on the most important CCP module and explains how the game processing is realized with this module.

5. GENERAL DEVELOPMENT TASKS

The development of the game state processing in online games consists of several tasks, as shown in Figure 6. Regardless of developing a multiserver MMOG or a single-server, small-scale action game, the developer has to care about three *general tasks*, AoI management, game state processing, and data-structure design, when building the game on top of RTF. If the game uses multiple servers, then multiserver

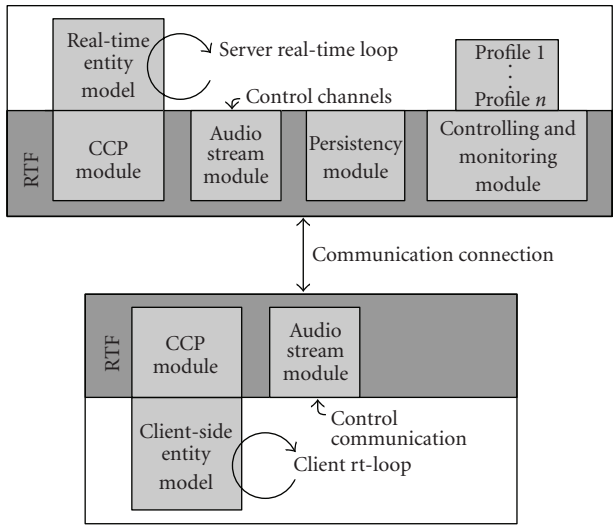


Figure 5: Modular approach of RTF.

parallelization and distribution also have to be taken care of by developers. Underneath these tasks for the developer, RTF provides a variety of low-level functionality like optimized event and entity serialization and communication, management of the game state and its possibly distributed processing. Overall, this separation of tasks among the developer and RTF reflects the overall approach of RTF sketched in Figure 2(e). Providing high-level game-engine-related functionality on top of an optimized communication middleware. The following subsections focus on the developer tasks and present the overall development methodology provided by RTF.

5.1. Task (1)—data structure design

The dynamic state of an online game is usually described as a set of *entities* representing avatars, NPCs, or items in the virtual world. Beside entities, *events* are the other important structure in an online game engine for representing user inputs and game world actions. Hierarchical data structures for events and entities in complex game worlds have to be serializable in an optimized manner for efficient network communication. When using only a communication middleware, developers have to build data structures and serialization mechanisms from scratch, while using an existing engine requires the use of predefined entities and events, which reduces flexibility.

RTF provides an optimized high-level entity and event concept enabling automatic serialization while still providing full design flexibility. When using RTF, entities and events are implemented as object-oriented C++ classes. The developer defines the semantics of the data structures according to the game logic. The only semantics of entities that is predetermined by RTF is the information about their position in the game world. Entities, therefore, are derived from a particular base class `Local` of RTF that defines the representation of a position for entities. This is necessary since the distribution of the game state processing across multiple servers is based upon the location of an entity in the game world. Besides the requirement of inheriting from `Local`, the design of the data structures is completely customizable to the particular game logic, as illustrated by the example of a racing car entity shown in Algorithm 1.

In order to enable platform independence, RTF defines primitive data types to be used (e.g., `rtf_int8`). Also, easy-to-use complex data types for vectors and collections are provided to the developer. Overall, more complex entity and event data structures can be easily defined using these primitives.

5.1.1. Automatic serialization and network transmission

In online games, entities and events are continuously transmitted over a network. Therefore, these hierarchical data structures have to be serialized in an optimized manner. However, there is no standard serialization mechanism in C++, such that the developer has to define and implement a network-transmittable representation for each entity and event of a game when using a traditional communication middleware. As an alternative, most engines provide high-level scripting capabilities with automatic serialization, but they decrease flexibility and possibly also performance due to the abstraction overhead from native C/C++.

RTF provides automatic and native serialization of the entities and events defined in C++, implements marshalling and unmarshalling of data types, and optimizes the bandwidth consumption of the messages. RTF solves this problem for the developer by providing a generic communication protocol implementation for all data structures following a special class hierarchy. All network-transmittable classes inherit from the base class `Serializable` of RTF. The `Serializable` interface can be (a) implemented by the developer, or (b) automatically implemented using the generic serialization mechanism provided by RTF. This automatic implementation is generated using convenient pre-processor macros provided by RTF. For all entities and events implemented in this manner, RTF automatically generates network-transmittable representations and uses them at runtime.

The generic serialization mechanism of RTF supports the following hierarchies:

(i) primitive data types (e.g., `float` or `std::string`);

(ii) serializable objects;

(iii) pointer to Serializable objects;

(iv) containers of Serializable pointers (e.g., std::vector),

(v) inheritances hierarchies of Serializables.

Serializable classes to be implemented with the generic serialization mechanism can have primitive data types as attributes. Serializable classes can be used for further derivation, for example, to form hierarchies of entities. Furthermore, classes derived from `Serializable` as well as pointers to Serializables can be used as attributes, see `_engine` or `_item` in Algorithm 1. The support for serializing a pointer to Serializable objects allows to realize

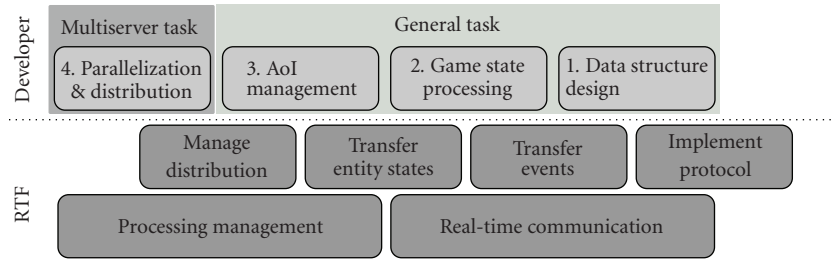


Figure 6: Development tasks for a multiserver game: distribution between RTF and developer.

```
class Engine        :public RTF::Serializable;
class Team          :public RTF::Serializable;
class PowerupItem   :public RTF::Local;

class MovingEntity  :RTF::Local{
    RTF::Vector   _velocity;
    RTF::Vector   _orientation;
};

class RacingCar     :public MovingEntity   {
    std::string    _driver;   // name of driver
    rtf_int8       _fuel;   // fuel level
    Engine         _engine;   // car's engine
    PowerupItem* _powerup;   // a specific item
    Team*     _team; // associated entity
};
```

Algorithm 1: An entity written in the manner of RTF.

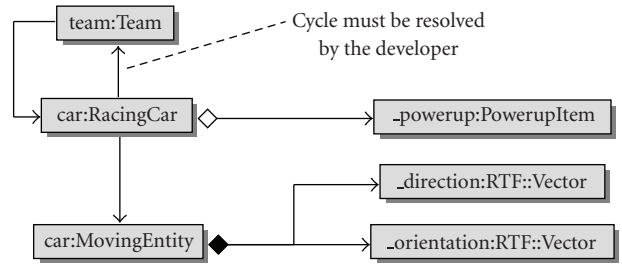


Figure 7: Example of an object graph supported by the generic serialization mechanism of RTF.

aggregation and associations in general. Support for using `Serializable` objects directly allows to map *composition*, that is, aggregation by value. It is also possible to use STL containers of `Serializable*` to express one-to-many associations. The number and depth of associations in the object graph is not limited by the generic serialization mechanism. However, a problem occurs when the object graph contains cycles. Such cycles are not automatically detected by the serialization mechanism, such that the developer must explicitly resolve them by denoting such associations as an *entity pointer*. An entity pointer is only serialized by the generic serialization mechanism as the ID of the referenced Serializable. For the developer of the entity hierarchy, it is not difficult to identify such associations. An object referenced as entity pointer needs to be treated like an entity: it must be manually subscribed to the AoI of a client, as described in Section 5.3.

The object graph shown in Figure 7 depicts associations for the example entity of Algorithm 1 and illustrates the object association treated by RTF's generic serialization mechanism. An entity of type `RacingCar` may have a `PowerupItem`, which will be serialized along with the `car` object. When serializing the `car` object, the generic serialization mechanism automatically deals also with the attributes of the `MovingEntity` base class. The graph contains a cycle as the example has a bidirectional association of `Team` and `RacingCar`—a car belongs to a certain `Team` whereas a `Team` is likely to have a list of cars. The `Team` itself needs to be available on the client independently from the player car. Therefore, it is easy for the developer to identify this fact and treat these associations as entity pointers, thus resolving the cycle.

For scenarios that forbid inheritance from the `Serializable` base class, the generic serialization mechanism provides the possibility to wrap entities within a generated, serializable version of themselves. This is, for example, necessary for the integration of RTF into an existing game engine with a specific inheritance structure. Although this wrapping imposes a slight code overhead to the default mechanism, it is still easy to use by developers and only has a minimal performance overhead.

Overall, this approach allows to use native C++ data structures for entities and events, while avoiding to implement the cumbersome, network-specific serialization by hand. Additionally, our approach is open to be combined with custom, engine-specific scripting capabilities, for example, LUA-bindings [9] for high-level behavior scripting can easily be added into the C++-based core data structures.

5.2. Task (2): game state processing

The central notion of our approach to the game development using RTF is the *real-time loop*, in which game states are updated. Most contemporary multiplayer games are based on such a loop, whose individual updates are called *ticks*. RTF allows the game developer, on the one hand, to implement his own real-time loop in the well-understood manner and, on the other hand, delivers him a substantial support for

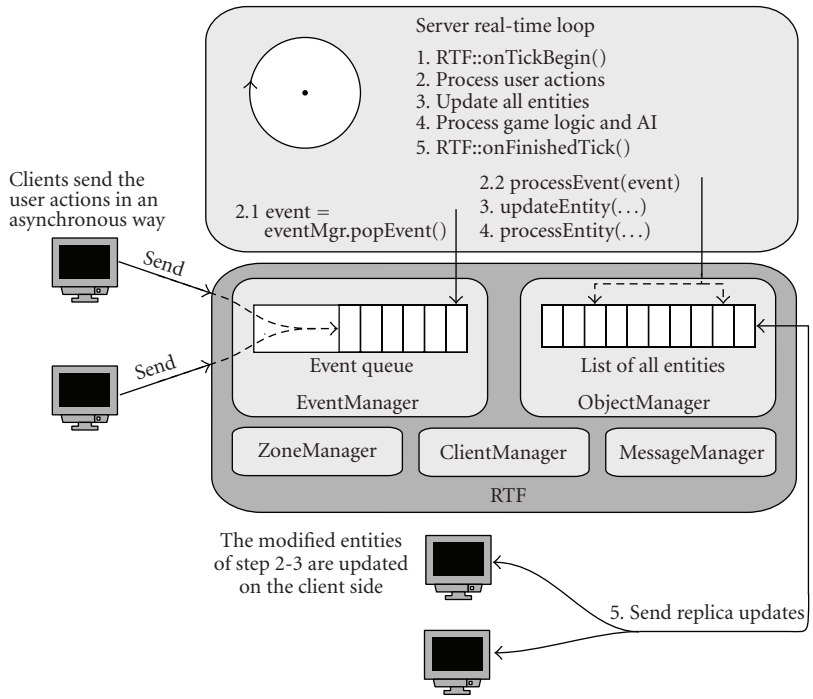


Figure 8: Server side game state processing integration with RTF.

implementing and running this loop on both the server- and client sides. RTF provides various manager classes accessible directly from his real-time loop as illustrated in Figure 8; these classes automatically manage several low-level aspects of an online game.

Figure 8 illustrates how the individual steps of the real-time loop are implemented by the game developer on the server side using RTF.

(1) First, the developer notifies RTF about the begin of a new tick by a call to `RTF::onTickBegin()`. Within this call, various low-level tasks are handled by RTF for the developer: incoming user actions are enqueued in the EventManager, new incoming client connections are dispatched and, if the game state is shared across multiple servers, then also game state updates from remote servers are applied automatically.

(2) The developer now processes the user actions received by the clients: the actions are retrieved from the EventManager, and the game state is updated as the reaction to the user actions according to the game rules. The ObjectManager keeps track of the game state, such that game entities are continuously added to or removed from the ObjectManager.

(3) In the third step, the entities are updated according to the game rules. In certain games, this may fall together with the next step.

(4) Artificial intelligence(AI), game logic and other update computations are performed. Some of these steps, like AI, might not happen in every tick.

(5) Finally, the developer notifies RTF about the end of the tick by a call `RTF::onFinishedTick()`. RTF executes most of its low-level runtime communication and distribution tasks within this call, including updates of the game state at remote clients. In the multiserver case, updates are also transferred to remote servers and modifications of the distributed game processing are handled; for example, connections to new servers are created and new servers are integrated into the game.

The real-time loop on the client side, illustrated in Figure 9, looks similar to the one on the server side, but works with a specific client side version. The developer has to perform the following steps in his client real-time loop.

(1) Similar to the server, a call to `RTF::onTickBegin()` is required at the begin of a tick. Incoming events sent by the server are enqueued and incoming game state updates are applied.

(2) The developer implements processing of the newly arrived server events, for example, incoming chat messages or notifications about in-game events.

(3) Typically, the server is not able to update the game state frequently enough to allow rendering of a fluent game progress. Therefore, game engines often use various interpolation and prediction techniques to compensate the low update rate of the server [10].

(4) The rendering step updates the visual screen of the player to show the updated game state. Also sound output and player input can be processed in this step.

(5) The developer notifies RTF about the end of the tick by a call `RTF::onFinishedTick()`.

This is the basic structure of the client real-time loop, although particular game designs may exclude some tasks from the real-time loop. Our approach works well with such

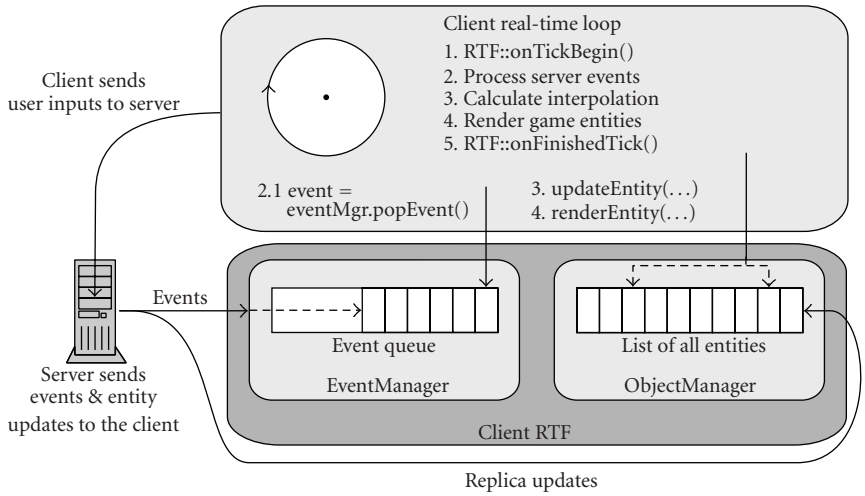


Figure 9: Client side game state processing integration with RTF.

games too, because loop-external tasks are synchronized with the game state which is still managed within the real-time loop.

This schema of integrating a communication and distribution middleware into the continuous processing of the game state is an important finding of our studies on providing a distribution middleware for online games: it frees the developer from low-level network programming which is required when using a conventional communication middleware. At the same time, our solution still provides full design flexibility for the real-time loop as opposed to approaches that use an existing game engine with a predefined processing loop.

5.3. Task (3): AoI management

An *area of interest (AoI)* concept assigns each avatar in the game world a specific area where dynamic game information is relevant and thus has to be transmitted to the avatar client. AoI optimizes network bandwidth by omitting irrelevant information in the communication. If done in a fine-granular manner, it avoids wallhack-like cheating [11, 12] at the client side which makes walls semitransparent and reveals hidden opponents outside of the AoI. Unfortunately, determining the relevant set of entities for a particular client can be quite compute intense, such that the AoI management, for which different algorithms are compared in [13], has to be implemented in an efficient and optimized manner.

RTF supports the custom implementation of arbitrary AoI concepts by offering a generic publish/subscribe interface for interentity visibility. The engine determines continuously which entity is relevant for a client avatar and notifies RTF of each change of an "interested" relation through a `client.subscribe(···)` and `client.unsubscribe(···)` call. RTF automatically takes care that the entity is available and always updated at the client or is removed from the client, respectively. RTF also takes care that entities are removed from the AoI of all participating clients if an entity disappears at a certain server as a result of this entity's movement from one zone into another and thus may be leave the AoI of clients implicitly.

5.3.1. Transferring entity states

Every time the game engine has finished the processing of a new game state, the RTF automatically synchronizes the state of entities between the distributed processes depending on the indicated AoI relations. When an entity is replicated to another process (e.g., all the entities within the AoI of a particular avatar to its client), the state of the remote copy has to be updated. Since the computations are usually performed in repeatedly executed cycles (*ticks*), the best way to perform state updates is after a computation cycle has finished, thus preventing propagation of intermediate states and read-write conflicts between the middleware and game engine.

The use of RTF simplifies this task of transferring entity states for the developer. He only has to inform the middleware that a computation cycle of the game engine has ended by invoking `RTF::onFinishedTick()` (step 5 in Figure 8). The necessary flow of information to update the game state on all participating processes is determined by the RTF upon the specified distribution. At runtime, the middleware automatically creates messages for changed objects and transmits them. This is done using the network transmittable representations that were generated for the data structures by the RTF preprocessor macros during the development cycle. The RTF part of the receiving process of such an update message automatically determines the object related to the messages and writes the updated data directly to the right object. Since the data is directly written to the objects used inside the game engine, this writing step is again triggered by the developer, for example, directly before a

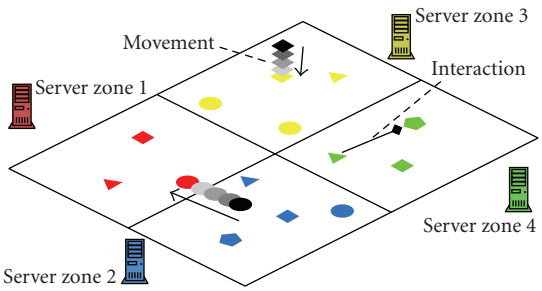


Figure 10: A zoned world distributed on four servers.

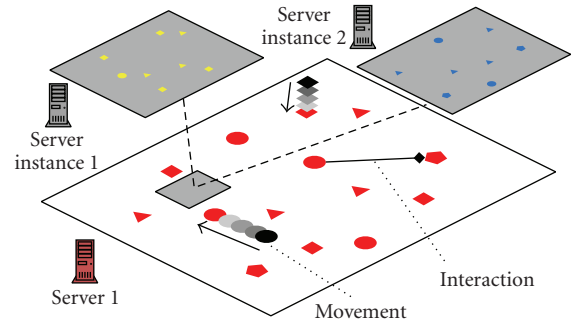


Figure 11: A virtual world with one instanced area processed independently by two servers.

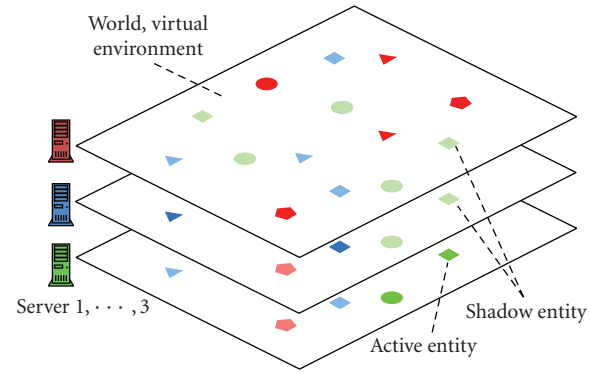


Figure 12: A zone that is cooperatively processed by three servers.

computation cycle, by invoking `RTF:onTickBegin()` (step 1 in Figure 8), to prevent read-write conflicts.

6. MULTISERVER TASKS

The general development tasks described in Section 5 are fundamental for any client-server-based game. However, in the case of massively multiplayer games, a multiserver approach is required for achieving high scalability. This section describes parallelization approaches supported by RTF and discusses how developers can easily use them for building MMO worlds.

6.1. *Parallelization concepts supported by RTF*

RTF currently supports three parallelization concepts and their free combination for scaling virtual world environments: *zoning*, *instancing*, and *replication*.

Zoning [14–16] partitions the virtual world into disjoint parts, called zones, and assigns each zone to one server. Figure 10 shows a virtual world with four zones on four servers and the clients and entities that move and interact within these zones. Although clients can move between zones, no interaction between clients in different zones is possible. Zoning is commonly used in contemporary MMORPGs like *EVE Online* [17] and *World of Warcraft* [18] and allows large player numbers in such games. The zoning approach fits best for games where the players are reasonably distributed in a very large virtual world.

The zoning concept requires that clients are always connected to their responsible server, that is, the one processing the client's zone. RTF performs run-time checks for all clients if this condition is met and transfers clients automatically to their responsible server. Such a transfer is completely transparent on the client-side RTF and only causes a notification on affected servers.

Instancing creates multiple copies of special parts of the game world. Figure 11 shows how a small area (the grey rectangle) is processed in separate copies by two different servers.

Instancing in online games can come in two flavors: complete zones can have several independent copies, which can be accessed by any player, and players that enter an instanced zone have to choose one particular copy. This type is usually not very appreciated by players, because it destroys the illusion of playing in a single world. The second flavor is to have instances for smaller areas in the game world and an instance is created simply by players, or groups of players, entering such an area. Such an area could be a very exciting dungeon and if a player enters it, he gets his own copy for the time he is in the dungeon. The copy is destroyed when the player leaves the dungeon. This second flavor is heavily used by MMORPGs.

Both flavors of instancing are supported within RTF. A client that enters an instanced area—depending on the instance flavor—either triggers an automatic instance creation within RTF or an existing instance must be specified as the transfer target. Subsequently, the client is automatically transferred by RTF to the server that is responsible for the new or the specified instance.

Replication [4, 19] is an alternative parallelization approach recently discussed in academia. Figure 12 shows three servers which cooperatively process the same virtual world zone. Each of the servers replicates his data in a symmetric manner and each server is responsible for some part of the overall data.

RTF supports the replication concept as it allows to add entities to a zone that is managed by multiple servers. A server which creates a new entity in a zone is automatically the responsible server for the entity which is called *active entity*. RTF automatically starts to replicate an active entity on all the remaining servers of the zone and these replicas are called *shadow entities* on the remaining servers.

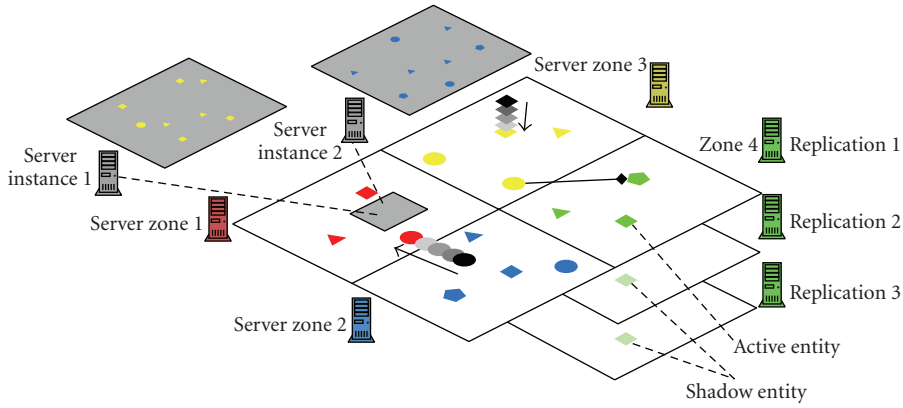


Figure 13: Combination of zoning, instancing, and replication for a single-game world in RTF.

All three orthogonal parallelization concepts aim at different scalability dimensions: zoning allows large user numbers in large MMORPG worlds, instancing runs a large number of game world areas independently in parallel, and replication targets high user density for action- and player-versus-player-oriented games. If the players are regularly distributed across the game world, then zoning is the best choice as it scales linearly with the number of zones. But if players tend to crowd at certain places and thus increase the player density at these places, instancing or replication is necessary. Instancing again scales linearly with the number of instances but introduces multiple copies and their conceptual drawbacks. If the creation of multiple copies of certain areas does not fit into the game concept, then only replication allows to scale the player density while providing a single, seamless world. The game concept usually determines the best possible combination of these concepts. For example, zoning could be used for a huge game world, while certain dungeons in this world are instanced for groups of players and cities are replicated to allow an increased player density.

A novel feature of RTF is the possibility to arbitrarily combine the three orthogonal distribution approaches depending on the requirements of a particular game design. Figure 13 illustrates a possible combination of these approaches in a single game as provided by RTF. This is an improvement as compared to MMORPGs which already combine zoning and instancing, but replication is currently not available for a combination with either of them.

The integration of all three concepts enables RTF to provide new manageability functionalities for games on RTF without a need for introducing a specific application support for these functionalities. Section 7 explains how the combination of zoning and replication enables RTF to transfer clients from one zone into another zone without a noticeable interruption on the client side. This combination also natively enables interactions across zone boarders, which must otherwise be implemented separately. The combination of instances and replication allows the reassignment of zones and instances during run time to new machines and the reaction on load increases in certain zones or instances. Adding an additional server to a zone or instance raises the supported number of clients for this zone or instance.

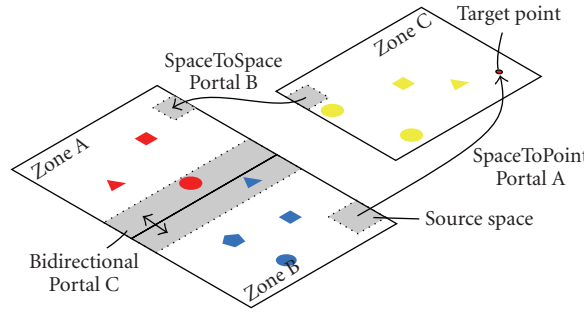
The overall goal of integrating these approaches into RTF, as discussed in detail in [20], is to provide general and dynamic scalability for all game genres within a single framework, which can be operated on demand in a grid computing environment. The following discussion sketches the envisaged methodology for developers for using these multiserver parallelization concepts.

6.2. Task (4): parallelization and distribution

If the multiserver capabilities of RTF are used, then, in addition to the general tasks 1–3 (data structure design, state processing, and AoI management), the developer has to segment the game world into zones, instances, and replication areas and to define the connections between them in form of portals. Using this information, RTF automatically assigns servers to each of the segments and connects each client to the particular segment the associated avatar resides in. If the user moves his avatar through a portal area, RTF will recognize this and automatically issue a connection transfer, making the server of the new segment responsible for processing the avatar. Each of the participating servers runs the normal server real-time loop discussed in the previous section for its associated segment—RTF internally handles connection migration and distributed entity management.

6.2.1. Specify segmentation

RTF offers a dedicated interface for specifying how the overall game world is segmented into a combination of zones, instances, and/or replication areas. A *world*-loader creates the specification with this interface once during application startup and the specification is then static for the overall application session. We provide a default loader which uses game world definitions in XML files, but developers may implement their own loader and file formats. Figure 14(a) illustrates a two-dimensional game world example with three zones and portals of various types. The definition of a zone,



(a)

```
Zone A = Zone (0,   Space (0, 2, 0, 4, 2, 0),   PLAIN );
Zone B = Zone (1,   Space (0, 0, 0, 4, 2, 0),   PLAIN );
Zone C = Zone (2,   Space (5.5, 1, 0, 4, 2, 0),   REPLICATE );
Portal & pA = *new SpaceToPointPortal ( entranceA, destinationA );
Portal & pB = *new SpaceToSpacePortal ( entranceB, destinationB );
Portal & pC = *new BidirectionalPortal ( spaceOne, spaceTwo );
```

(b)

Figure 14: Segmentation specification example.

as illustrated in Figure 14(b), consists of an ID, the occupied space, and a flag if it is allowed to replicate the zone across multiple servers. For the portals, an entrance area and a connected destination area are given. During runtime, all zones are assigned to the set of available servers. Figure 14(b) shows different portal types supported by RTF for expressing various transfer relations (uni- bidirectional, space to space, space to point) and how they connect different areas of the game world.

6.2.2. *Implement segment-related game logic*

With the introduction of zones, replications and instances—the segmentation of the game world—also the game-world-related update processing should be segmented. For example, a server should place new NPCs only in the zone he is responsible for and he should only create in-game events that are related to this zone.

For the realization of necessary interzone and interserver events, RTF provides three mechanisms: a server can send events and messages to a certain zone; a server can send events to the owner of a certain shadow entity; and a server can create global objects. If an event is sent to a certain zone, RTF automatically determines and transmits the event to the responsible servers (could be multiple servers for a replicated or instanced zone). If an event is sent to the owner of a shadow entity, RTF determines the server that holds the corresponding active entity and transmits the event to this server. Finally, global objects are serializable objects that are replicated automatically to all participating servers of the game. A global object can contain, for example, the information of the current weather for the complete game world or a global scoreboard. RTF currently does not provide a distributed synchronization mechanism for the collaborative modification of global objects and currently the global object owner serializes the distributed write access. Efficient synchronization mechanisms which are appropriate for real-time online games are being investigated and will be incorporated into upcoming RTF versions.

6.2.3. *React upon distribution changes*

As clients and entities can move between zones and instances, the game-state distribution may change during runtime. Therefore, the developer is informed about clients and entities that have entered or left a zone. If a client or entity moves into a certain zone, the responsible server is notified about this event and must process the updates and events related to this client or entity.

Overall, game developers only have to implement mechanisms at a high level of abstraction in the RTF multiserver task. In particular, they can start developing any multiserver game engine as a single-server engine at begin and then easily switch over to a scalable multiserver engine. For this switch, developers, in most cases, only have to segment the game world into zones, instances, and replication areas, possibly implementing segment-related game logic mechanisms on top of the already existing specified entity and event data structures.

7. SEAMLESS GAME WORLD AND ZONE MIGRATION

Using zoning as a distribution concept is subject to two restrictions: (a) entities and clients must be transferred between the participating servers if they are moved between the zones, and (b) no interactions are allowed across zone borders. Traditionally, the game developer implements the transfer functionality by explicitly establishing a connection to the new server and communicating the entity view from this server to the client. To allow interactions, for example, attacking a remote entity across the zone border, special synchronization and interserver communication are

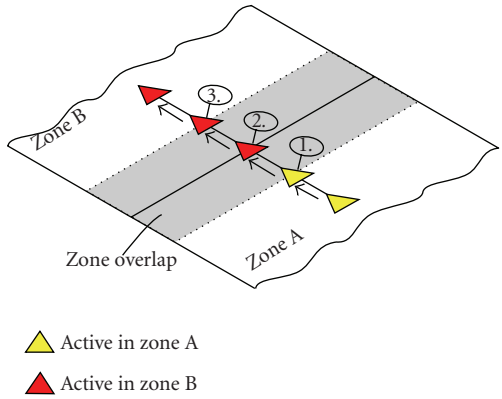


Figure 15: Overlapping zones for a seamless migration of an entity between two zones.

required, increasing therefore the overall complexity of the game architecture and reducing its scalability.

In this section, we describe how RTF provides a transparent solution for these problems and, furthermore, allows to move zones between servers. RTF allows a seamless interzone migration and interaction by creating an overlapping area between two or more adjacent zones. Since RTF allows to freely combine zoning with replication, this overlap is replicated across the servers.

Figure 15 illustrates how two zones are overlapped in the 2D case, thus creating a seamless game world. The bottom part of the figure demonstrates how the movement of an entity between these zones is handled.

(1) The entity moves within zone A into the overlap and is replicated as a shadow entity on the server of zone B (step 1 in the figure).

(2) After the entity covers half the distance of the overlap, RTF automatically changes its status in A from active to shadow and vice versa in B (step 2).

(3) As soon as the entity leaves the overlap, RTF removes it from zone A (step 3).

If the entity is a client's avatar, then the communication connection of the client to the server of zone A must be transferred during step 2 to the server of zone B. RTF manages this seamlessly if the developer makes the overlap bigger than the area of interest of the client: in this case, both servers responsible for A and B have the same view of the game world within the replicated area, such that no initial communication between the new server and the client is necessary. Furthermore, interactions across the two zones are now possible because they take place within the replicated overlap area and the client is placed in both overlapping zones at the same time. In summary, this leads to a completely seamless game world for the clients.

RTF also supports a migration of a zone to another server during runtime, by using a replication mechanism similar to the one shown in Figure 15. Since the migration is performed over an extended period of time, no interrupts are necessary, such that the players observe a smooth game flow. In addition, this allows to dynamically assign servers depending on the current system load and maintenance work, which is particularly interesting for the utilization of grid resources.

8. RTF IN GRID CONTEXT

RTF integrates solutions which enable games on top RTF to be easily deployable and executable in a grid system. Its interface realizes an abstraction of the participating resources, for example, events are sent by a developer to zones instead of particular hosts. This abstraction allows RTF to add hosts and relocate zones to different hosts transparently to the game developer. Furthermore, RTF provides an integrated in-application monitoring and controlling module, allowing to manage an application on top of RTF by external management consoles which connect to this module.

The work on RTF is part of the *edutain@grid* project funded by the EC IST, where it provides the fundamental real-time computation and communication middleware for interactive applications and online games operated in a grid computing infrastructure. It is developed with a strong emphasis on studying and optimizing mechanisms in the area of distributed real-time computation and communication, continuous processing parallelization and development methodology of distributed virtual environments and online games. Games implemented on RTF and operated in a grid environment can be easily and automatically adapted to changing user demands. Possible scenarios currently taken into account in edutain@grid include the following.

(1) *Daytime-dependent user load:* At prime time at night, each zone can be operated by a dedicated server for maximum performance, while a single server can be responsible for several zones during daytimes with low demand. This frees resources for different tasks or allows to efficiently share a pool of resources among several games or sessions at a data centre.

(2) *User hot spots:* Users of games with large worlds like MMORPG tend to cluster in particular zones for socializing, trading, or fighting with each other at large scale. This dynamic behavior leaves some zones nearly empty, while the hot spot zones may be congested. The replication approach allows to dedicate additional server resources to such a heavily frequented zone, thus scaling the maximum player density in that area for maintaining a fluent game experience. This replication can be dynamic: if users move over to an adjacent zone, the grid environment can automatically remove replication servers form the old zone and assign them to the new heavily utilized zone.

(3) *Server role change:* Especially in MMORPGs, a lot of large raids for player versus environment (PvE) gameplay often form at night and enter dungeon instances for fighting large boss mobs for several hours. This behavior implies a large demand for instance servers during that time, while at other times of the day instances might be barely frequented. The edutain@grid system is designed to switch the roles of RTF-based game servers: in this example, unutilized replication servers could be switched to become instance

servers during main raiding time, and be switched back to replication servers if only few instances are requested.

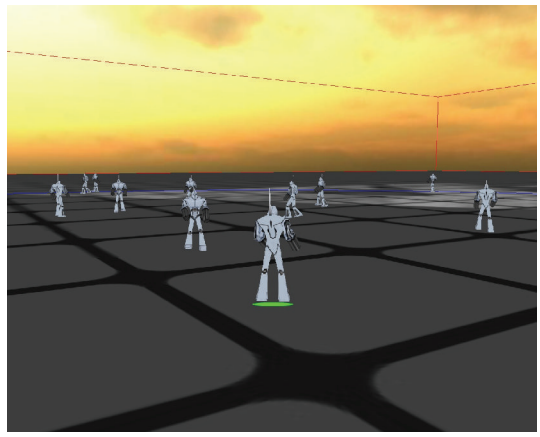
Although RTF is developed with the goal in mind to support these scenarios and to ease the development of grid-enabled interactive online applications in cooperation with grid management services, its major focus is to be a development tool on its own. RTF is used in the first place by developers to realize their online games on a high level of abstraction while RTF cares about the efficient serialization, communication, and distribution management of the game state and processing. The manageability features that RTF can provide beneath this high-level abstraction are particularly interesting for the dynamic usage of cluster- or grid resources for the game service provisioning.

9. IMPLEMENTATION CASE STUDIES

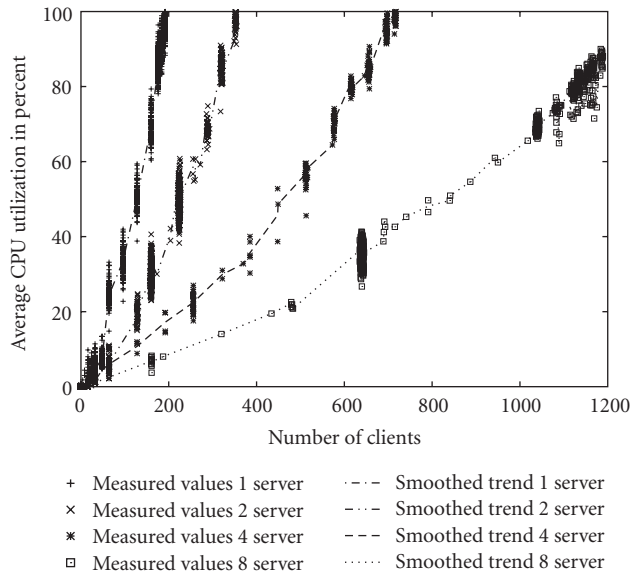
Several case studies are currently developed on top of the RTF prototype. The *RTFDemo* application is a fast-paced evaluation and demonstration game that takes place in a zoned 3D world. The game state is updated 25 times per second and the zones are overlapped and allow a seamless migration across zone boarders. Each client has control over a single avatar and can move it around and let it interact with the game world. Figure 16(a) shows a screenshot of one client in the game, looking at avatars of other clients. We used a heterogeneous setup of PCs with 2.66 GHz, CoreDuo 2 CPUs, and 4GB RAM in a LAN setup for preliminary performance and scalability tests. The average CPU utilization was measured as a metric for the test and multiple computer-controlled clients continuously sent inputs to their server. Clients could move freely between zones, but were initially distributed equally between the servers. Figure 16(b) shows the number of players that could participate fluently in the game for one, two, four, and eight zones. The current RTF version already achieves more than 160 clients on a single server with a high update rate of 25 Hz. RTF also meets the expectation that the zoning approach should scale nearly linearly if the clients are equally distributed.

For an evaluation of the overall development process and methodology based on RTF, we are developing an MMOG named *Offshore*, which takes place in an aquatic metropolis. Figure 17(a) illustrates the corresponding game world segmented into nine zones, while Figure 17(b) shows a screenshot of the current client prototype giving an overview of the game world from an elevated position.

A custom game engine, relying on RTF, was built for this MMOG by 12 developers in six-month, part-time student project and incorporates Ogre3D as a rendering and input engine, OgreAL for sound, and TinyXML for map loading. The integration of all components into a custom game engine went very well and all basic elements of a faster-paced MMORPG are present. RTF first supported the general development tasks for single-server operation, after which the game engine has been successfully switched over to multiserver processing by segmenting the game world. The prototype has about 58 K *lines of code* (LoC) whereas RTF itself has 35 K LoC and we estimate that the usage of RTF has saved about 25 K LoC in the prototype. This project mainly served as a test how understandable the development methodology of RTF is to nonexpert developers and the feedback was very positive and helped to improve the developer manual.



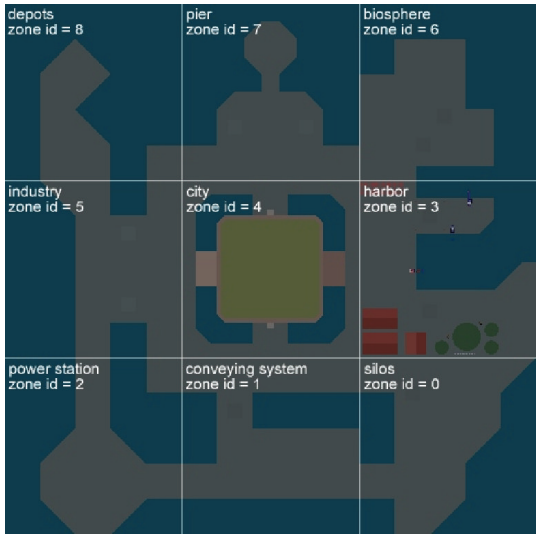
(a) In-game screenshot of RTFDemo game



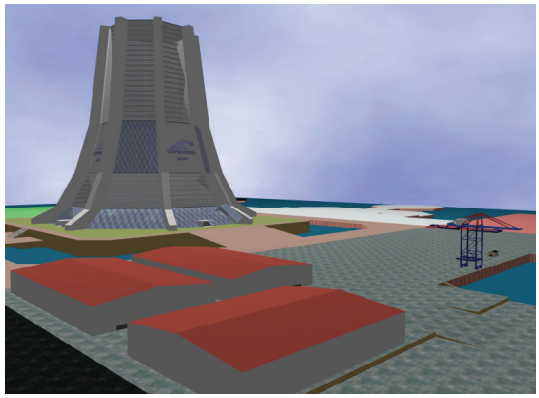
(b) CPU utilization for different numbers of servers

Figure 16: RTFDemo: the fast-paced evaluation game and the results of the experiment.

For an evaluation of the responsiveness of RTF, its flexibility and suitability for fully fledged fast-paced games, the publicly available *Quake 3* game engine is currently ported onto RTF. Quake 3, illustrated by the screenshot in Figure 18(a), is a very popular first-person shooter which is highly optimized for performance and low network traffic. RTF replaces the network module of the engine and the current beta state already allows to run Quake 3 on top RTF. Technically, Quake 3 is originally written in plain C and it was necessary to make Quake 3 C++ compliant, as RTF is designed and implemented in C++. Along the RTF integration, Quake 3 is also modified to support



(a) Offshore metropolis segmentation



(b) Overview screenshot of current prototype

Figure 17: Offshore—a basic MMOG.

the replication distribution concept. Although Quake 3 is extended for replication, the substitution of the original network module with RTF reduced the code amount from about 328 K LoC to 313 K LoC. A detailed report on this work of porting Quake 3 onto RTF is currently in preparation for publication.

Beside the case studies that we are developing, external partners incorporate RTF into their applications. Darkworks [21], a French game-development studio, is working on the integration of RTF into an upcoming professional game engine. The University of Linz incorporates RTF into their *Net'O'Drom* [22, 23] racing game. The game now uses a zoned world with seamless migration between the zones. Figure 18(b) shows a screenshot of the Net'O'Drom game from an elevated position.

Overall, these case studies and external developers provide a lot of valuable feedback. The successful and easy integration of RTF into all these projects supports the position that the RTF development process and methodology are suitable for a large class of online multiplayer games and especially complex multiserver online games.

(a) Screenshot of the quake 3 port

(b) Screenshot of the Net'O'Drom port

Figure 18: Existing games that are ported onto RTF.

10. CONCLUSION AND RELATED WORK

In this paper, we have studied and analyzed contemporary development methods for massively multiplayer online games and demonstrated to what extent the low-level custom development can be substituted by a high-level approach using game middleware for single-server and for scalable multiserver engines. Our particular new contributions are as follows.

(1) We provide a comprehensive taxonomy of contemporary game development approaches with respect to their flexibility and level of abstraction. Based on this taxonomy, we describe a new development approach that aims both at single-server and multiserver settings and still provides a very high degree of flexibility. Game developers are liberated from the low-level communication and distribution management tasks while being able to realize the remaining game development tasks without inappropriate restrictions.

(2) We describe in detail our RTF middleware system which is used to support the human developer in the development process. RTF's comprehensive distribution capabilities enable a smooth transition from a single-server to a multiserver game design. Since RTF focuses on the processing part of games, it puts no constraints on the remaining development tasks as, for example, graphics or game logic implementation.

(3) We sketch how RTF automatically enables the dynamic usage of grid resources for changing user demands

and describe common scenarios where the dynamic exploitation of grid resources could be interesting.

Zoning [3] and replication [4, 19] were already investigated successfully as independent distribution concepts for scalable multiplayer online games. Our high-level development approach integrates both concepts, including instancing, in a seamless way and we described and tested RTF as a development tool for this high-level approach.

Also various high-level game development approaches have been proposed, for example, [24, 25] investigate the abstraction of the overall game engine, making it possible to exchange an underlying game engine without modifying the game. This development approach represents an even higher level of abstraction, compared to ours, but requires the implementations for various network-related issues, graphics-rendering, input-processing, and so on. In our approach, RTF supports an easy realization of these issues in the context of online games. Project Darkstar [26] proposes a separation of all game-related processing into task objects that are freely distributable across multiple servers. An underlying run-time and global object store distributes the tasks and manages the distributed object access. However, a global object store and random distributed access might be difficult to manage efficiently in a very responsive and highly interactive game. Also [27, 28] present development approaches for multiplayer games. However, [27] discussed their novel high-level approach so far only for peer-to-peer-based or single-server-based online games and [28] focuses on monolithic, compile-time management of the complexity of an MMOG-architecture, while we provide a more incremental development approach with strong focus on runtime functionality of our RTF middleware.

In comparison to existing approaches in the field of basic communication middleware like *Net-Z* [6], *HawkNL* [29], or *RakNet* [30], RTF provides a much higher level of abstraction: this includes automatic entity serialization and hides nearly all of the technical network communication aspects. On the other hand, RTF is significantly more flexible than reusable game engines like the *Quake* or *Unreal* engines, because it is not bound to a specific graphics engine and leaves the real-time loop implementation to the developer, who is now supported by the high-level mechanisms of RTF for entity and event handling. The multiserver capability of RTF allows to easily incorporate three different parallelization and distribution approaches and is open to be extended in future game designs. This flexible support of different parallelization concepts allows RTF to be usable for a broader range of MMOG concepts than existing multiserver middleware like the Emergent Server Engine [7] or BigWorld [8] which focus mostly on the concept of zoning.

Our proposed high-level development approach is efficiently supported by the current RTF implementation, which both provides a high level of abstraction and preserves design flexibility for single- and multiserver game engines. We conducted several case studies which showed that RTF is indeed easy to use and it successfully shields the developer from the low-level tasks of online game implementation.

Summarizing, RTF offers the following integrated functionalities.

(1) Game data structures are specified as plain C++ entities. The serialization for the transfer over a network is done automatically by RTF, and the underlying communication implementation is optimized with delta updates to reduce the amount of data sent over the network.

(2) The game logic and entities are implemented in a usual object-oriented way and are open to be integrated with state-of-the-art scripting capabilities, like LUA [9], for example.

(3) The proven multiserver distribution concepts zoning, instancing, and replication, as well as their combinations, are supported by RTF. The corresponding segmentation and distribution of the game world are described on an abstract level.

(4) Distribution management and parallelization of the game state processing is fully handled by RTF. Segments can be reallocated to new servers during runtime and interserver client migrations are realized in a seamless way.

(5) Support for advanced monitoring and controlling capabilities simplifies the dynamic management of online games in a grid environment with resource management.

Besides the in-depth evaluation of RTF's performance characteristics, we are investigating the applicability of our approach to broader, nongame classes of applications that still exhibit a basic real-time loop structure, for example, e-learning and spatial physics simulation. Furthermore, we plan to integrate additional features into RTF in the future. In particular, additional grid-related monitoring and manageability functions are highly promising to be integrated for further enhancing RTF as a comprehensive middleware for online games.

ACKNOWLEDGMENTS

The authors like to thank the anonymous reviewers, whose comments helped them a lot to improve this paper. The work described in this paper is supported in part by the European Union through the IST 034601 project "edutain@grid."

REFERENCES

[1] F. Glinka, A. Ploss, J. Müller-Iden, and S. Gorlatch, "RTF: a real-time framework for developing scalable multiplayer online games," in *Proceedings of the 6th ACM SIGCOMM Workshop on Network and System Support for Games (NetGames '07)*, pp. 81–86, Melbourne, Australia, September 2007.

[2] A. Ploss, F. Glinka, S. Gorlatch, and J. Müller-Iden, "Towards a high-level design approach for multi-server online games," in *Prooceedings of the 8th International Conference on Intelligent Games and Simulation (GAMEON '07)*, pp. 10–17, Bologna, Italy, November 2007.

[3] M. Assiotis and V. Tzanov, "A distributed architecture for MMORPG," in *Proceedings of 5th ACM SIGCOMM Workshop on Network and System Support for Games (NetGames '06)*, ACM, Singapore, October 2006.

[4] J. Müller-Iden, *Replication-based scalable parallelization of virtual environments*, Ph.D. thesis, Universität Münster, Münster, Germany, July 2007.

[5] R. Prodan, V. Nae, T. Fahringer, et al., "Toward a grid environment for real-time multiplayer online games," in *Proceedings of the CoreGRID Integration Workshop (CGIW '08)*, Crete, Greece, April 2008.

[6] Quazal net-z, 2006, http://www.quazal.com/.

[7] Emergent Game Tech., 2007, http://www.emergent.net/.

[8] BigWorld, "BigWorld Technology," http://www.bigworldtech.com/.

[9] W. Celes, L. H. de Figueiredo, and R. Ireusalimschy, "Binding c/c++ objects to lua," in *Game Programming Gems 6*, M. Dickheiser, Ed., pp. 341–355, Charles River Media, Rockland, Mass, USA, 2006.

[10] J. Smed and H. Hakonen, *Algorithms and Networking for Computer Games*, John Wiley & Sons, New York, NY, USA, 2006.

[11] J. Yan and B. Randell, "A systematic classification of cheating in online games," in *Proceedings of 4th ACM SIGCOMM Workshop on Network and System Support for Games (NetGames '05)*, pp. 1–9, ACM, Hawthorne, NY, USA, October 2005.

[12] C. Choo, "Understanding cheating in Counterstrike," November 2001, http://www.fragnetics.com/articles/cscheat/print.html.

[13] J.-S. Boulanger, J. Kienzle, and C. Verbrugge, "Comparing interest management algorithms for massively multiplayer games," in *Proceedings of 5th ACM SIGCOMM Workshop on Network and System Support for Games (NetGames '06)*, ACM, Singapore, October 2006.

[14] W. Cai, P. Xavier, S. J. Turner, and B.-S. Lee, "A scalable architecture for supporting interactive games on the internet," in *Proceedings of the 16th Workshop on Parallel and Distributed Simulation (PADS '02)*, pp. 60–67, IEEE, Washington, DC, USA, May 2002.

[15] P. Rosedale and C. Ondrejka, "Enabling player-created online worlds with grid computing and streaming," September 2003, http://www.gamasutra.com/resource_guide/20030916/rosedale_01.shtml.

[16] M. R. Macedonia, M. J. Zyda, D. R. Pratt, P. T. Barham, and S. Zeswitz, "NPSNET: a network software architecture for large-scale virtual environments," *Presence*, vol. 3, no. 4, pp. 265–287, 1994.

[17] CCP. EVE, http://www.eve-online.com/.

[18] World of Warcraft, http://www.worldofwarcraft.com/.

[19] A. Bharambe, J. Pang, and S. Seshan, "Colyseus: a distributed architecture for online multiplayer games," in *Proceedings of the 3rd Symposium on Networked Systems Design & Implementation (NSDI '06)*, pp. 155–168, USENIX Association, San Jose, Calif, USA, May 2006.

[20] J. Müller and S. Gorlatch, "Scaling online games on the grid," in *Proceedings of the 4th International Game Design and Technology Workshop (GDTW '06)*, M. Merabti, N. Lee, K. Perlin, and A. El Rhalibi, Eds., pp. 6–10, Liverpool John Moores University, Liverpool, UK, November 2006.

[21] Darkworks s.a, 2008, http://www.darkworks.com/.

[22] C. Anthes, A. Wilhelm, R. Landertshamer, H. Bressler, and J. Volkert, "Net'O'Drom—an example for the development of networked immersive VR applications," in *Proceedings of the 7th International Conference on Computational Science (ICCS '07)*, pp. 752–759, Springer, Beijing, China, May 2007.

[23] H. Bressler, R. Landertshamer, C. Anthes, and J. Volkert, "An efficient physics engine for virtual worlds," in *Proceedings for the Medi@terra Art & Technology Festival*, pp. 152–158, Athens, Greece, October 2006.

[24] A. BinSubaih and S. C. Maddock, "Game portability using a service-oriented approach," *International Journal of Computer Games Technology*, vol. 2008, Article ID 378485, 7 pages, 2008.

[25] A. BinSubaih, S. C. Maddock, and D. Romano, "A survey of 'game' portability," Tech. Rep. CS-07-05, University of Sheffeld, Sheffield, UK, 2007.

[26] Inc. Sun Microsystems, Project darkstar, http://www.projectdarkstar.com/.

[27] P. Kabus and A. P. Buchmann, "A framework for network-agnostic mutliplayer games," in *Proceedings of the 8th International Conference on Intelligent Games and Simulation (GAMEON '07)*, Bologna, Italy, November 2007.

[28] V. Narayanasamy, K.-W. Wong, and C. C. Fung, "Complex systems-based high-level architecture for massively multiplayer games," in *Game Programming Gems 6*, M. Dickheiser, Ed., pp. 607–622, Charles River Media, Rockland, Mass, USA, 2006.

[29] Hawk Software, HawkNL, 2006, http://www.hawksoft.com/.

[30] Rakkarsoft, RakNet, 2003, http://www.rakkarsoft.com/.

Hindawi Publishing Corporation
International Journal of Computer Games Technology
Volume 2008, Article ID 594313, 9 pages
doi:10.1155/2008/594313

Research Article

ALVIC versus the Internet: Redesigning a Networked Virtual Environment Architecture

Peter Quax, Jeroen Dierckx, Bart Cornelissen, and Wim Lamotte

Expertise Center For Digital Media, tUL, IBBT, Hasselt University, Wetenschapspark 2, 3590 Diepenbeek, Belgium

Correspondence should be addressed to Peter Quax, peter.quax@uhasselt.be

Received 31 January 2008; Revised 25 April 2008; Accepted 16 June 2008

Recommended by Jouni Smed

The explosive growth of the number of applications based on networked virtual environment technology, both games and virtual communities, shows that these types of applications have become commonplace in a short period of time. However, from a research point of view, the inherent weaknesses in their architectures are quickly exposed. The Architecture for Large-Scale Virtual Interactive Communities (ALVICs) was originally developed to serve as a generic framework to deploy networked virtual environment applications on the Internet. While it has been shown to effectively scale to the numbers originally put forward, our findings have shown that, on a real-life network, such as the Internet, several drawbacks will not be overcome in the near future. It is, therefore, that we have recently started with the development of ALVIC-NG, which, while incorporating the findings from our previous research, makes several improvements on the original version, making it suitable for deployment on the Internet as it exists today.

1. INTRODUCTION

Witness the media attention that applications such as Second Life [1] and There [2] have gathered, it should be obvious that the technology behind them is an important subject of study. Indeed, historically, many architectures have been proposed, that were designed from the ground up to be scalable. In modern day's terms, however, the figures that can be attained are not exactly state of the art. Looking at examples of [3] or [4], the authors point out that architectures scale up to a maximum of tens of users. Comparing this to a modern-day application such as Second Life, which claims to have around 1.3 million active residents (a subject of debate), or World of Warcraft, with around 10 million active subscribers, these numbers indeed do seem ridiculously low.

Once the architecture behind these success stories is exposed, however, it quickly becomes clear how the developers have tweaked the systems in such a way that the user is tricked into imagining the virtual world being a single-instance massive environment. In practice, World of Warcraft, for example, uses a system of instancing where a limited amount of players are active in a single instance. By adding additional instances of the same virtual world, the community of players can grow indefinitely (as long as servers are added). However, it is clearly not possible for all players in the community to interact—as they need to be active in the same instance. A different approach is taken by Second Life, a single-instance virtual world, where the virtual land is split up into several regions, each managed by a single server. While this approach is definitely simple to implement, there are very obvious limitations, such as the (possibly disastrous) growth of network and processing load on the server once an event takes place in a location that is deemed popular by the community. Several circumventions have been implemented to mitigate these problems, for example, the limitation of the number of polygons that can make up a model in the environment, or the splitting of regions that have become too popular. This latter solution, however, leads to additional costs in terms of server infrastructure and cannot cope with a highly dynamic world, as the region definition, in terms of server assignment, is relatively static.

2. RELATED WORK

Several years ago, we investigated an alternative architecture, able to support the same numbers of users that are needed for todays' applications, that is, the Architecture for Large-Scale Virtual Interactive communities (ALVICs) [5]. ALVIC was designed from the ground up to be scalable to high numbers of users, all present in a single instance of the virtual world. The basics behind ALVIC were founded on several (at the time) next-generation network features that were thought to become available in the near future. It has turned out, however, that the promised improvements are clearly not yet met. This is the reason why a new version of ALVIC is being designed (dubbed ALVIC-NG), that takes into account the limitations apparent in today's version of the global Internet architecture. For reasons of clarity, a brief description of the original ALVIC architecture is given in Section 3, we do, however, refer the reader interested in more details to our previous work published in [5, 6].

There are some commercially available products (and architectures) that show some familiarities with our work. For example, the BigWorld [7] middleware platform is claimed to be the upcoming industry standard by its developers. The software collection that is offered consists of a set of server applications, together with a 3D client and specially developed API's. Unfortunately, as is common with most commercially available products, the technical details are not disclosed; it is, however, clear that a client/server architecture forms the basis of BigWorld. The technology behind World of Warcraft [8] may seem—at first glance—to be able to support millions of simultaneous users. In practice, however, the World of Warcraft system is based on a sharded design, whereby multiple instances of the virtual world (called shards or realms) are run concurrently on a large number of servers. This means that only a small subset of players is—at any given time—able to interact with each other, typically a few thousand. The architecture also supports "instancing," through which a group of players (typically less than 25 players) can indicate that they wish to complete a quest without interference from other players. The World of Warcraft architecture is fundamentally different from the ALVIC-NG architecture in the fact that it uses sever realms to support the total number of players that is subscribed. In case the number of players outgrows the server capacity, a new realm is started on a new server cluster. Readers familiar with the Eve On-line [9] architecture may notice several parallels with the approach used for that particular game. We will point out, as part of the discussion on the new architecture, some key differences, although the intricate details about the implementation of the EVE Online architecture are also not publicly available.

When comparing ALVIC-NG to Second Life [1], one of the best-known and most successful (single-shard) 3D virtual communities, we should remark that Second Life uses a fixed assignment of (virtual) geographical regions to servers. At the time of writing, a maximum of about 35000 simultaneous users were active in the virtual world, which is run on more than 5000 servers, each serving an area of 256 by 256 meters. While this type of design is easy to implement, scalability problems are sure to become apparent as soon as the number of users increases. As users are not typically evenly distributed over the virtual world, some servers are nearly idle, while some are overloaded. The ALVIC-NG architecture is designed to use the available server processing power as efficiently as possible, thereby decreasing the chance of system failure in case a large number of users decide to convene in a single location in the virtual world.

There are also some architecture-only solutions that should be compared to our ALVIC-NG framework. One of these is the Sun Game Server Technology framework [10]. This is fundamentally different from our approach, as the virtual world is not spatially subdivided; every server is able to manage each object through a massive centralized database. For each operation that is to be performed on an object, the information is retrieved from the database, and stored again after the manipulation is completed. While this is clearly scalable in terms of the size of the virtual world, this solution also introduces extra delay for each operation that is to be performed on an object (which may accumulate if interactions involve several objects). This architecture is designed to be scalable up to around 10000 users, and it uses a cluster of database servers that are load-balanced. A second example of a similar architecture is the Multiverse technology platform, which also does not use a "traditional" spatial subdivision scheme for scalability purposes (it does in fact have such a scheme for visibility purposes, but this is rather trivially implemented and not relevant here), but rather tries to scale the number of supported users by defining services that are implemented through plugins. Examples of these services are those that handle combat events, intricate interactions, and so forth. At run-time, those plugins that are able to support a large number of users can be migrated to servers that are minimally loaded. However, it should be noted that some plugins are inherently more processor-intensive than others (or are sure to be used much more than others), so they will probably be assigned to the most powerful servers anyway.

In research, several other architectures have been proposed and discussed, which were designed to support networked virtual environment applications, both client/server and peer-to-peer based. In this section, we have specifically opted to discuss only those that are currently being used in the specific context of games. We do refer the reader to [6, 11, 12] for a comprehensive overview of existing literature on the more general subject of NVEs.

Section 3 provides a brief overview of the original ALVIC architecture, which is required to understand some of the design options that were made for ALVIC-NG. Section 4 describes the problems associated with ALVIC, when one wants to deploy the architecture on real-life networks such as the Internet. In-between solutions that can be used to overcome some of the limitations are presented in Section 5. The next generation of ALVIC is described in Section 6. Conclusions and pointers to future work are presented in Sections 7 and 8.

3. THE ORIGINAL ALVIC ARCHITECTURE

The architecture behind ALVIC was designed to be adaptable to several usage scenario's, ranging from games to virtual interactive communities. Each of these applications should be able to be deployed on the same architecture, preferably even concurrently. However, in practice, this means that each end-user can, at a given time, only be present in one specific world. Because of the extensive size of the virtual world, we followed an approach similar to that in [13], in which each virtual world, running on the server infrastructure, is divided into a number of square regions. Their size depends on the estimated number of active clients in that region and on the type of region. For example, a region that represents a small room inside a building would most likely be scaled to equal the dimensions of the room. Clients that move around the world dynamically enter and leave regions depending on their position.

The reasoning behind this subdivision of the world is to effectively link the physical properties of the virtual world (geographic location) with the underlying network architecture. The relation between the two entities is strong because of the fact that data propagation can easily be coupled to visibility. If an object is invisible to the end-user, there is no need for any data to be received. Furthermore, by assigning a distinctive multicast address to each of the regions defined before, we can reduce unnecessary network traffic.

In fact, event information, originating from a single end-user, should only be sent to the multicast address of the region from which the event originated. When a client enters a region, a simple subscription to the multicast group assigned to that specific region suffices to start receiving state information on all objects present in the region. As all members of a region send their generated events to the same multicast address, it should be clear that they will also receive all events from other members in the same group without the need for an explicit distribution mechanism through a dedicated or ad hoc-defined server.

It should be clear that a mapping of these (geographical) regions onto multicast groups is an efficient way of distributing data. There is no need to maintain open connections with a "number of" server(s) to receive state information. Neither would one need to determine where to send data, as the current location is always known by a client. The key to the entire system is the fact that data distribution within a multicast group is done implicitly.

Besides this first trivial task, each client is responsible for managing its own *area of interest* (AOI), analogous to, for example, the work in [14]. It is of vital importance to note (as stated before) that there is a coupling between geographical regions and their associated multicast addresses. It can clearly be seen that at a specific moment in time, a limited number of other regions will be located in the view frustum of a client. It is, therefore, only necessary for a client to subscribe to exactly those regions. The view frustum size is entirely client-side determined, and can be adapted dynamically to either expand or shrink depending on several factors, such as available bandwidth or processing power. We point out here that a large view frustum does not have any impact on the upstream traffic needed for sending out state information, as this data only needs to be sent to the local multicast address.

While, in theory, it is entirely possible to design a networked virtual environment architecture using only multicast traffic, we opted to include a set of governing servers into the architecture. Their purpose is threefold: authentication, network resource management (e.g., multicast addresses), and server resource management. The minimal load on these servers in the architecture allows for a large number of clients to be simultaneously connected to a single server (for more details, see our previous work, as referred in Section 2) and facilitates the distribution of load over several physical machines.

4. ALVIC-SPECIFIC DEPLOYMENT ISSUES

While ALVIC has been shown to scale to several thousands of users using only a very limited number of servers (see [15]), the features it relies on to make this possible have still not become widely available on the Internet as it is available to typical end-users. In this section, we will identify the main issues that still exist, and will remain problematic in the near future. While some intermediary solutions exist for some of the problems, a good example of this is the use of TURN for NAT traversal, they cannot always be applied to the specific transmission methods used in ALVIC (i.e., multicast transmissions). Also, not all solutions provide satisfactory results due to the special requirements posed on the multicast transmissions employed by ALVIC (e.g., IGMP snooping would introduce prohibitive amounts of delay, IPv4 tunneling of IPv6 traffic is rather inefficient, etc.). These restrictions have been the main reasoning behing the development of the next-generation ALVIC architecture.

4.1. IPv6 deployment

When ALVIC was first proposed, the mass introduction of IPv6 was touted as being the solution to many of the problems facing the Internet community at the time. Several features, such as a massive increase in the machine-addressing space and the support for large numbers of multicast addresses (together with improved supporting protocols), would make it possible to manage a large set of multicast groups, necessary for ALVIC to be deployed for massive environments. It has turned out, however, that while the backbone networks of ISP's do support or actually run on IPv6, the availability of this technology to typical end-users is still severely limited, and will probably remain so for the next few years.

4.2. NAT gateways

The main reason behind this is the proliferation of IPv4 NAT gateways and firewalls, which mitigate the problems associated with the limited number of addresses available. By hiding several machines behind a common IP address, these machines are at the same time able to connect to the Internet, and they are (relatively) protected against attacks from the

outside. For ALVIC, however, NAT gateways present a major obstacle, as direct peer-to-peer connections are required for the architecture to work as it was originally designed. Even worse, the support for multicast applications behind NAT gateways is practically nonexistent. To provide an optimal experience to the end-user, it is clearly undesirable that major reconfiguration of network equipment (such as port forwarding and/or definition of DMZs) is required to run an application.

4.3. Multicast address space and scope

Multicast applications, as they exist today, are based around a very limited set of content producers that distribute their data to large amounts of "listeners." This is especially true for the case of digital TV distribution (possibly on-demand), where a single producer (the broadcasting company and/or network provider) sources all data watched by subscribers. As these networks are managed by a single entity (the network provider), the scope of the multicast transmissions can be limited to the provider-owned network. For ALVIC purposes, addresses with a global scope are clearly required, as a single instance of the virtual world is required for all users. Coming back to the example of Digital TV, it should also be clear that only a limited number of addresses is required (e.g., one for each stream in a set of TV channels). ALVIC, on the other hand, requires large amounts of multicast group addresses, if it is to be deployed with a fine-grained spatial subdivision scheme. At the same time, real-life wide-area networks are optimized specifically for one-to-many multicast applications. However, in the case of ALVIC, users generate their own multicast traffic, which needs to be transmitted from their own computers or devices to the other participants, something which cannot be done on these types of networks due to the possible explosive growth in traffic.

5. INTERMEDIATE SOLUTIONS

As it became clear during the final stages of the development of ALVIC that the Internet would not quickly evolve in the direction that was required for deployment, a temporary solution was envisaged that would overcome several practical issues, while retaining the advantages a multicast-based architecture could offer.

We used the CastGate [16] project, which, in practice, consists of two entities. One of these entities, the router, is placed in the local, multicast-enabled LAN. Its role is to intercept the packages that are to be transmitted to the multicast-enabled backbone network. The link between the router and the multicast-enabled backbone network is unicast only. A separate entity needs to be placed in the backbone network as an end point for the tunnel between the different networks.

Using this approach, we were able to interconnect several sites using a number of routers, every one of them connected to the multicast-enabled BELNET network. While it would, in theory, be feasible to have each of the connected parties install this additional piece of software and to reconfigure their network equipment, this is clearly undesirable from a user's point of view. It is, therefore, only interesting for academic reasons.

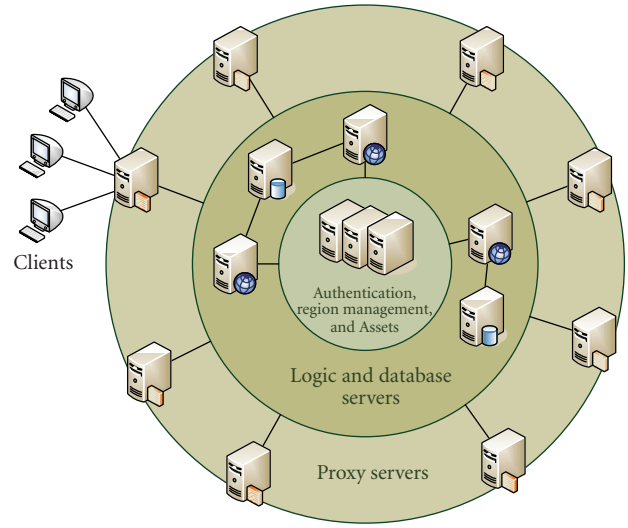


Figure 1: Conceptual overview of ALVIC-NG.

6. ALVIC-NG

The limitations as described above, combined with the unsatisfactory intermediate solution described in Section 5, have triggered a major redesign of the ALVIC framework, now designated as ALVIC-NG (next generation).

We have been very careful to retain the strong points of ALVIC, while translating them into a more real-life network-friendly architecture, mainly based on the client-server paradigm instead of peer-to-peer.

6.1. Overall overview

The main entities of the architecture are shown in Figure 1, represented in a set of concentric circles. At the outer perimeter, the clients are shown that want to connect to the virtual world. Instead of connecting to a variety of supporting servers as is often the case in current-generation examples, such as Second Life, nearly all traffic is tunneled over the client-proxy link. The proxies are responsible for handling a number of clients at the same time, and are assigned based on several properties. These may include, for example, their processing load and/or the network properties of the link between the client and the proxy (e.g., typical RTT values and/or packet loss). More on this subject can be found in next section. Proxies are assigned from a pool of available servers, managed by a centralized entity, which is also responsible for other authentication and accounting tasks. This entity, as mentioned in what follows, is based on the master server of the original ALVIC architecture.

As the spatial subdivision and area of interest management scheme used in ALVIC provided us with a powerful way to manage downstream bandwidth, we wanted to retain this system for the new architecture. However, the peer-to-peer approach needed to be substituted with a client-server-based

equivalent. The new entities responsible for managing parts of the world are referred to as Logic servers. They are notably different from, for example, the simulators in the Second Life architecture, in the way they are assigned to geographical regions in the virtual world. Instead of using a fixed allocation scheme, as is traditionally used, a new entity is created, responsible for managing the relationship between virtual locations and Logic servers, called the Region Management (RM) system. Analogies can easily be drawn between these RM servers and the DNS system that is currently in use on the Internet. The RM system can be queried by the proxy servers to find out which region(s) is/are managed by a specific server. At the same time, the RM system is responsible for keeping track of the load on the various logic servers. A control link is, therefore, established between the RM system and the individual logic servers, over which several parameters are sent, comparable to the SNMP querying system. In case the RM system detects either a Logic server failure or an impending overload of a specific server, the Logic servers are re-assigned to remediate the problems. Possible solutions include splitting the management of a single part of the virtual world over a number of servers or transferring the complete responsibility to a new instance, for example, in case of complete logic server failure. A more detailed scenario is described in Section 6.2. When compared to the original ALVIC design, the Region Management system is roughly comparable to the Game server entities.

Logic servers can also be used as entities that control the behavior of objects, such as non-playable characters (NPC's) or autonomous interactive objects such as virtual video walls. Behaviors are triggered by scripts that are assigned to specific objects. As the logic servers are responsible for handling all objects present in a specific part of the virtual world, which will traverse the virtual word, the scripts need to be shared between all logic servers. These scripts, together with the information regarding the visual representation of objects, are stored in asset databases.

The reason behind the introduction of the intermediary layer of proxy servers in the architecture is threefold. First of all, it reduces the number of connections each client needs to initiate and maintain with other servers (which may lead to issues as discussed in Section 4 due to the presence of firewalls and NAT gateways). Secondly, the proxies reduce the number of connections for the logic servers, which is important if a high number of clients is to be supported on a single machine due to the overhead associated with connection tracking. Finally, the proxies can "cache" a lot of data, possibly reducing the response time (and load) on the logic servers, as these servers can be assigned in such way that they provide a better response time than the entire path between the client and the logic server(s).

As with any virtual environment system, persistent storage is a requirement to keep the world up and running over long periods of time. It also offers enhanced features such as roll-back capabilities in case of system failure and/or, more applicable to the virtual world scenario, in case of malevolent users that have exploited the system. Instead of using a single, high capacity database, as is typical in existing applications (e.g., Eve Online), the ALVIC-NG architecture provides a fine-grained mechanism for determining the degree of persistency that is required. In case transactions being handled have financial repercussions (e.g., the exchange of virtual currency between users), it is likely that these transactions need to be logged and written to disk immediately, as an in-between state, where currency is "floating" between users would clearly not be desirable. However, it should be clear that not all objects and actions require an immediate storage of state to disk. This enables the ALVIC-NG architecture to retain as much state as possible in the main memory of the Logic servers, which improves both response time and the load on the database servers. It is, in any case, the goal of ALVIC-NG not to use a single server (farm) for persistent storage, as the requirements on this type of server would increase in a nonlinear fashion with a growing number of users. A clear demonstration of this fact is the limitation that is put on the objects that are present in a single simulation server (analogous to our logic servers) in Second Life. In reality, only about 15000 prims (primitive objects such as spheres, cones, etc.) can be supported on a single server [17]. Also, relational database systems are CPU-intensive applications. An example is given in [18], where a cluster of more than one hundred machines is required to support about 30000 transactions per second in a game context. The persistency modules of ALVIC-NG are designed in such a way that they can employ a number of distinct servers, again based on factors such as load or network link capacity. As the update rate of the database system is low due to the in-memory processing and adaptive storage requirements for different classes of operations (e.g., player movement versus financial transactions), we are able to use a basic MySQL infrastructure, were a number of instances of this software can run on the same hardware as the logic servers. Persistency and the inclusion of logic servers that do processing on parts of the environment is something that is entirely new to the ALVIC-NG framework, as the (previous) peer-to-peer approach necessitated the individual clients to be responsible for the distribution of their own state information. In case the client would disconnect, there was no way to store his/her data in a central location. Please note that ALVIC-NG only provides an interface to a back-end, which will in most cases consist of an off-the-shelf database management system (either object-oriented, relational, or any other type). A benchmark for the performance of this back-end is outside the scope of this paper. The back-end should provide functionality such as roll-back capabilities and redundant storage of information, which will automatically provide the ALVIC-NG framework with the same capabilities.

6.2. *Typical usage scenario*

To clarify the interdependence of the various entities in the architecture, we will describe the typical workflow for a client that connects to the system and subsequently moves around in the virtual world. We refer to Figure 2 for a graphical representation.

Initially, the client is unaware of the existence of the proxy and logic servers. The only publicly known entities

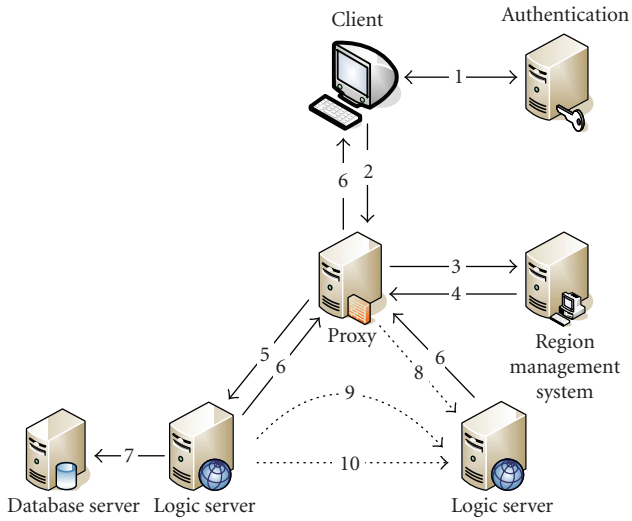


Figure 2: Typical usage scenario of ALVIC-NG.

are the Authentication servers. A (possibly minimal) pool of proxy servers is available, and the members of this pool are known by the Authentication servers (through the region management facility, which was described in the previous section). Once a connection has been established by the client (1), a specific proxy is assigned. The decision on which proxy to select is based on several metrics, including the current "load" on the proxy. In practice, the load calculation on the proxy servers is achieved through several parameters, including the number of clients it is currently serving, as well as the 5-minute-average CPU load (which is available, e.g., on linux systems through the /proc file system). At regular intervals, this load information is exchanged between the proxy servers and the region management infrastructure, enabling the authentication servers to choose a proxy server that currently has a minimal amount of "load." Of course, a proxy server may still decide to reject clients based on momentary information, available only to the server itself (after which the process repeats from the beginning). Beside the load factor, another metric is used in determining the appropriate proxy server to assign, namely, the network delay between the client and the server. The network delay is sampled from (existing) active connections between clients and proxies, and is also communicated through the region management infrastructure. If this data is not available (in case of a newly introduced proxy server), the geographic location of the proxy server can be compared to that of the client (available through, e.g., a WHOIS database lookup) and used as as intuitive metric. While this does not guarantee an optimal assignment, it does severely decrease the chance that a server is chosen on a very impaired network path. In practice, a combination of several metrics can be used to provide satisfactory results.

Once authentication is finished, the client is redirected to the proxy server and establishes a (reliable) control connection (2). At the same time, a UDP data channel is established by sending out datagrams to a specific port on the server (dynamically assigned). This enables the packets to be sent in the reverse direction, possibly traversing the NAT gateway at client side.

Subsequently, the client will announce its initial position to the proxy server it is connected to. We should point out here that only the proxy servers are aware of the assignment of regions to servers, not the clients. This enables regular updates of the mapping, without requiring notifications to be sent to all connected clients. The starting position of the client determines its starting region, and the associated Logic server address is determined (by the proxy) by querying the Region Management servers (3 and 4). If the client is located in a region for which the proxy server does not yet have an active connection to the logic server, the connection is established (5) and the client position is announced to the Logic server in question. At the same time, all updates originating from the specified region are, through the proxy server, sent to the client (6). This may also include additional regions, as required by the AOI management scheme applied.

On a regular basis, determined by the persistency requirements as explained above, the data stored in the Logic server's memory is stored in one of the many database servers (7).

If a client is moving around in the virtual world, and traverses a region boundary, the proxy will detect this, request the associated Logic server address by querying the RM system, and connect to the new Logic server (8). The information state associated with the client is removed from the old server and uploaded to the new one (9).

At any given time, the RM system may determine that one of the Logic servers is overloaded and/or has failed. In the former case, a migration of data will take place, called a region split. This involves storing the Logic server's state in a database and/or directly exchanging information between servers (10) (depending on connectivity) and subsequently assigning the newly created regions to the appointed logic servers. If required, it is possible for the new Logic servers to update their state memory by reading it from the persistent storage medium. In case of a region split, the region boundaries will be updated, and these updates will be announced to the proxy servers (the clients remain unaware of the world buildup). An analogous scenario can be envisaged for the merger of two regions with minimal client load. In case of server failure, a new server is assigned and the state is recreated from what is available in persistent storage.

The decision of splitting and merging regions (at runtime) is left to the region management system. As we stated before, the load on several entities in the architecture is communicated in specific intervals between these entities and the RMS. In this case, the logic servers gather "local" information on the amount of regions they are currently managing, the amount of open connections to proxy servers and the amount of objects in memory (or persistent storage). This information is compared to static information on the available processing/handling power of each of the logic servers (determined by link capacities and raw CPU power). Based on these metrics, a decision will be taken to either split or merge regions if the server becomes overloaded or even superfluous (due to client inactivity in specific regions). However, a single strategy cannot be cited as being the "best"

solution under all circumstances. Depending on the type of game or overall player behavior, it may or may not be desirable to have frequent updates in the spatial subdivision scheme. The ALVIC-NG architecture does not depend on a single metric to determine region management, but is rather developed in such a way that new metrics can easily be added. We will come back to this issues when discussing some of the simulation results in Section 6.4.

6.3. Advantages of ALVIC-NG

In this section, we will look at how the new ALVIC-NG architecture overcomes the issues described in Section 4.

First of all, tunneling all traffic required for a session through the proxy servers enables us to have a severely limited number of open connections and streams at any given time. They also remain the same during an entire session, which is an ideal situation when considering NAT gateways and firewalls. All TCP connections can be initiated at client side, and UDP sessions can easily be kept alive as the port numbers remain the same. There is no need for any incoming peer-to-peer traffic that may be blocked by the network configuration.

The spatial subdivision scheme, proposed in ALVIC, was retained but redesigned to be independent of multicasting capabilities of the network. We should, however, point out here, that the software architecture underlying ALVIC-NG is designed in such a way that the previous implementation using multicast is still supported. The fact that regions are now assigned to Logic servers instead of multicast groups in a dynamic way enables us to exchange the data using unicast connections, albeit with the additional overhead caused by this distribution method. To mitigate this clear disadvantage, the proxy servers are also able to act as caching servers, and distribute "known" data to users without having to fetch the data for each client individually.

Using a (possibly large) set of proxy and logic servers, globally distributed, relieves the need for multicast addresses with a global scope. At the same time, it enables optimal connection circumstances for clients (in terms of delay and link capacity), without the additional delays associated with the propagation of IGMP messages required for multicast traffic.

Of course, there is a tradeoff when switching to a client-server-based architecture from a peer-to-peer approach. For one, we loose the automatic distribution mechanisms offered by multicast transmissions. At the same time, the additional investment in terms of server infrastructure may be a hindrance to the uptake of applications based on ALVIC-NG. We do feel, however, that the added advantages, in terms of being able to be deployed on any current-generation broadband access network technology, as well as the ease of management and moderation outweighs these disadvantages.

6.4. Simulation and results

To test the concepts introduced in the ALVIC-NG architecture, we have implemented the various elements to serve as a test-bed for scalability tests. The applications are deployed on a dedicated 16-node PC cluster, interconnected through a gigabit LAN. For testing purposes, the "client" application was developed with a dual interface: one that uses 2D/3D visualization (to confirm the correct functioning through the eyes of a user) and a command-line version; the latter enables us to deploy a large number of instances of the client software on a single machine. As static clients (in other words, clients that do not move around in the virtual world) are not representative of real-life users, a scripting language (LUA [19]) is used to move the avatars in the virtual world, based on predefined movement patterns. As the behavior is scripted, and scripts are assigned on a random basis, the result is a semirandom population of the virtual world. The client with visualization enabled allows us to check whether the system works as intended. Besides the normal client functionality, this version also is able to query the servers in the world to get an overall outlook on the region assignment to logic servers.

A sample is shown in Figure 3, where the world is divided into seven regions. We should point out here that, for these simulation results, we have opted to manage the virtual world as a quad-tree, as this is an efficient data structure for fast detection of boundary passing and can easily be split/merged. The ALVIC-NG architecture, in principle, can be extended to work with a generic region definition. The "active" avatar is shown by the blue dot, surrounded by the circle indicating its area of interest. The "active" regions are indicated by the slightly brighter colors (in this case, these are the ones that overlap with the clients' AOI). The other avatars, which are, as stated before, steered by scripting, of which state information is being received, are also visualized. By moving around in the virtual world, this simulation allows us to test that handovers between logic servers can be handled without major delays and impact on the user experience. It should be noted that a certain delay cannot be avoided, as there is a propagation delay for the new data to arrive at the proxy. The simulation setup will enable us to effectively determine worst-case figures for this delay value and examine how this deficiency can be masked (e.g., through tweaks in the graphical rendering engine). At the same time, it allows us to test the efficiency of the region splitting/merging algorithm that is implemented, which may depend on a number of metrics, as mentioned in Section 6.2. The application also allows us to force the split/merge operation on regions to simplify the testing process. In Figure 4, another scenario is visualized, in which a more intricately subdivided world is shown, together with a client with a reduced AOI (indicated by the smaller surrounding circle).

7. FUTURE WORK

The load and scalability tests on the ALVIC-NG framework are ongoing work. Based on our findings, the metrics used for determining the optimal time to split/merge regions can be adapted. The ALVIC-NG architecture is to be used as a basis for story-telling applications, games, and community-related features in the IBBT Teleon Project, the goal of which

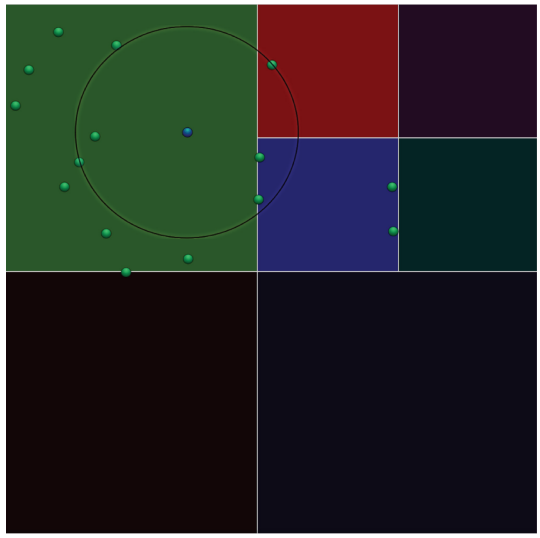


Figure 3: Sample spatial subdivision based on a quad-tree. Active regions are brightly colored.

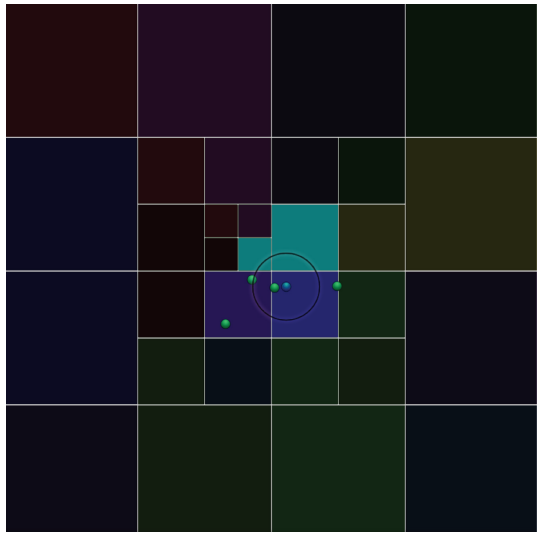


Figure 4: A client with reduced AOI in a more intricately subdivided virtual world (distributed over several logic servers).

is to deploy the architecture as the base for a nation-wide platform (under supervision of the Flemish radio and TV broadcasting company VRT).

8. CONCLUSIONS

In this paper, we have introduced ALVIC-NG, a generic framework supporting networked virtual environment applications. The technology presented is applicable to both games and virtual community applications. Based on our original findings during development of the ALVIC architecture, the NG version is designed from the ground up to be deployable on heterogeneous networks, independent of the availability of next-generation network features such as user-generated multicast data distribution and large numbers of globally scoped multicast addresses. The unique selling points of ALVIC, in terms of its spatial subdivision scheme and scalability, have been retained in a novel, mainly client-server paradigm-based architecture. Besides purely scalability-related features, ALVIC-NG also offers solutions for issues that plague many current-generation applications, such as intricate NAT traversal and the additional problems associated with peer-to-peer traffic flows.

ACKNOWLEDGMENTS

Part of this research is funded by the European Fund for Regional Development (EFRD). The authors are grateful to the partners involved in the IBBT Teleon project, as well as the members of the NVE research group at EDM.

REFERENCES

[1] BigWorld Technology, *BigWorld*, http://www.bigworldtech.com/.

[2] Blizzard, *World of Warcraft*, http://www.worldofwarcraft.com/.

[3] MULTIVERSE, *Multiverse*, http://www.multiverse.net/.

[4] P. Quax, *An architecture for large-scale virtual interactive communities*, Ph.D. thesis, Transnationale Universiteit Limburg, Limburg, Belgium, 2007.

[5] K. L. Morse, "Interest management in large-scale distributed simulations," Irvine Technical Report TR 96-27, University of California, Berkeley, Calif, USA, 1996.

[6] P. Quax, T. Jehaes, P. Jorissen, and W. Lamotte, "A multi-user framework supporting video-based avatars," in *Proceedings of the 2nd Workshop on Network and System Support for Games*, pp. 137–147, ACM Press, Redwood City, Calif, USA, May 2003.

[7] Linden Labs, *Second Life Forums*, https://jira.secondlife.com/browse/MISC-210.

[8] C. Greenhalgh and S. Benford, "A multicast network architecture for large scale collaborative virtual environments," in *Proceedings of the 2nd European Conference on Multimedia Applications, Services and Techniques (ECMAST '97)*, pp. 113–128, Milan, Italy, May 1997.

[9] M. Capps, D. McGregor, D. Brutzman, and M. Zyda, "NPSNET-V: a new beginning for dynamically extensible virtual environments," *IEEE Computer Graphics and Applications*, vol. 20, no. 5, pp. 12–15, 2000.

[10] CastGate, *CastGate*, VUB ETRO-TELE, http://www.castgate.net/.

[11] There, *There.com*, http://www.there.com/.

[12] R. C. Waters, D. B. Anderson, J. W. Barrus, et al., "Diamond park and spline: a social virtual reality system with 3d animation, spoken interaction, and runtime modifiability," Tech. Rep. TR-96-02a, Mitsubishi Electric Research Laboratories, Cambridge, Mass, USA, November 1996.

[13] SUN, *Game server technology white paper*, Sun, http://www.sun.com/solutions/documents/white-papers/me_sungame-server.pdf.

[14] P. Quax, P. Monsieurs, T. Jehaes, and W. Lamotte, "Using autonomous avatars to simulate a large-scale multi-user networked virtual environment," in *Proceedings of the ACM SIGGRAPH International Conference on Virtual Reality Continuum and Its Applications in Industry (VRCAI '04)*, pp. 88–94, ACM Press, Singapore, June 2004.

[15] Microsoft, *SQL Server 2008 benchmarks*, http://www.microsoft.com/sqlserver/2008/en/us/benchmarks.aspx.
[16] C. Joslin, T. Di Giacomo, and N. Magnenat-Thalmann, "Collaborative virtual environments: from birth to standardization," *IEEE Communications Magazine*, vol. 42, no. 4, pp. 28–33, 2004.
[17] Linden Labs, *SecondLife*, 2003, http://www.secondlife.com/.
[18] LUA, *The LUA programming language*, http://www.lua.org/.
[19] M. Matijasevic, "A review of networked multi-user virtual environments," Tech. Rep. TR97-8-1, The Center for Advanced Computer Studies. Virtual Reality an Multimedia Laboratory. The University of Southwestern Louisiana, Lafayette, La, USA, 1997.

Hindawi Publishing Corporation
International Journal of Computer Games Technology
Volume 2008, Article ID 642314, 9 pages
doi:10.1155/2008/642314

Research Article

The Playing Session: Enhanced Playability for Mobile Gamers in Massive Metaverses

S. Cacciaguerra and G. D'Angelo

Department of Computer Science, University of Bologna, Mura A. Zamboni 7, 40127 Bologna, Italy

Correspondence should be addressed to G. D'Angelo, gda@cs.unibo.it

Received 31 January 2008; Revised 14 April 2008; Accepted 23 May 2008

Recommended by Jouni Smed

Internet ubiquity and the success of mobile gaming devices are increasing the interest in wireless access to virtual environments. Mainly due to the mobility factor and wireless medium features, traditional gaming architectures are not enough to guarantee good levels of playability and fairness to mobile gamers. We suggest a new mechanism, called playing session, capable of controlling communications between mobile devices and the game infrastructure. In case of network failures, a mimicking mechanism is in charge of playing, until the communication channel is restored. The goal is to reproduce, with an adequate level of mimesis, the user behavior. According to this approach, it will be possible to enhance the overall playability of Internet games without requiring any modification to the existing communication infrastructure.

1. INTRODUCTION

"*You take the blue pill, the story ends here, you wake up and believe whatever you want to believe. You take the red pill you stay in wonderland and I'll show you just how deep the rabbit hole goes.*" In this way, Morpheus offers Neo to be woken up by an illusory simulated reality, called Matrix, which is developed by intelligent machines in order to use human beings as their source of energy. Matrix is just one of the many visions describing the future Internet as a global cyberspace humans can explore and shape through their avatars. Words like Cyberspace, Metaverse, and Matrix are synonymies: all of them refer to a virtual reality-based evolution of Internet. In this scenario, the avatar is a tool allowing humans to interact with a metaverse (i.e., a meta multiuniverse). Over the last years, the interest of the gaming industry has led to the implementation of many metaverses called virtual worlds (VWs). Many of them are so realistic that they have an economy, government, and currency of their own (e.g., World of Warcraft [1], Second Life [2], Project Entropia [3], Sociolotron [4]). Thanks to the massive diffusion of wireless Internet access and to the increasing miniaturization of hardware devices, there is a growing interest in extending the massive online gaming to also nomadic users. Due to the unreliable nature of the wireless medium and to the mobility, this kind of gamers would require special mechanisms to maintain a good level of playability, at least from a technological point of view. In this work, we propose a new mechanism aiming at enhancing playability for all gamers (both wired and wireless). Our mechanism introduces a new level in the communication protocol stack that is in charge of controlling communications between clients and servers. The mobile nature of wireless gamers can be often the cause of interruptions and lags in the communications between devices and gaming infrastructure. We propose the playing session (PS) mechanism to enhance the gaming playability while maintaining good levels of equity and fairness between all users. The main task of PS is to monitor gaming actions and to quickly react in case of network failures, hence taking control of all avatars disconnected from the players (i.e., orphan avatars). The PS scope is not limited to network failures, it is triggered every time the gamer is unable to fulfill the deadline set by the game progress (i.e., hard-to-use input hardware interface of small devices or user disabilities) [5].

The reminder of the paper is organized as follows. Section 2 illustrates the problems when participating in massive metaverses from mobile devices. Section 3 introduces

related works. Section 4 highlights the main design issues to enhance the gamers' playability. Section 5 describes the proposed system architecture. Section 6 presents a case study: a clone of the Armagetron game. Finally, Section 7 concludes this work with some final remarks.

2. BACKGROUND

As seen in the intoduction, the wireless access to VW could be a big market success. It is a common assumption that, in the next years, Internet will be ubiquitously available, at least in some parts of the world. Furthermore, due to many practical and cost-related reasons, the last hop will be often based on wireless technologies. The combined effect of a widespread availability of Internet and the development of a new generation of wireless devices will lead to a massive amount of gamers interested in VW.

Today's portable devices integrate wireless network technologies into high-performance multimedia terminals, with the explicit aim at enabling distributed multiplayer gaming [6]. The potential of these devices is very high, but it is influenced by the characteristics of the underlying network technologies. For example, the playability of real-time multiplayer games is dominated by the end-to-end network latency [7]. In the case of 802.11 WLAN networks (Wi-Fi), clients should be able to communicate with an acceptable latency and data rate, while many problems are due to the movement of users. For example, what happens when a nomadic user wants to participate in an Internet game, but he is too far from all access points (APs)? In this scenario, packets coming from the mobile device and directed to the VW servers might be lost due to the lack of connectivity in the area or delayed due to the interaccess point handoff. The wireless scenario might generate many different situations: *horizontal handoffs* (it refers to the process of transferring a data session from one channel to another), *vertical handoffs* (it refers to a change in the technology, e.g., from Wi-Fi to UMTS), *interferences* (generating transmission errors), *closure* of the communication channel (e.g., deauthentication and disassociation). Moreover, many communication protocols were designed to comply with static nodes. What happens when a mobile device connects to an AP that belongs to a different internet service provider (ISP)? In this case, the mobile device should be able to obtain a new valid IP address. This could lead to an incorrect management of the network communications, and even, in a positive case, to the reconfiguration, while resume mechanisms might require many seconds to bring the system back to a working state. In order of importance, we classify the effects of the wireless communication faults: *short interruptions* due to the loss of some data (i.e., loss of packets, datagrams, or segments), *long interruptions* mainly due to the reconfiguration and the resume activities (i.e., protocol disconnections or application shutdowns), *permanent interruptions* mainly due to incorrect communication management at the application level (i.e., unexpected shutdown of the application, system crash). In this scenario, users playing from a mobile device and therefore using unreliable networks could be severely disadvantaged with respect to "wired gamers." They might lose some match turns (which is very unfair) and be also disconnected from the whole system. In all of these cases, the overall playability would be dramatically decreased.

3. RELATED WORK

A lot of studies focus on the communication-related problems arising from the distributed nature of the gaming architectures, both from the server and the client point of view [8]. That is, the impact of packet loss and communication latency on the playability and the fairness of Internet games have been widely investigated. Beigbeder et al. [9] have studied the effects of the packet loss and latency on user performance in "Unreal Tournament, 2003." In this case, the analysis is focused on, the so-called, first person shooters (FPSs), a class of games that is considered more sensitive than others to the changes in network performances. Instead, in [10] the effect of latency on users' performance has been inspected in case of a real-time strategy (RTS) game: Warcraft III. As expected, due to their nature, RTS games can tolerate a limited amount (less than a second) of latency without impacting on the overall outcome. Conversely, FPSs are greatly affected by latency: even a modest increase of the communication latency reflects in a deep degradation of the user performance. For the sake of simplicity, the proposed solutions can be divided into two main approaches:

(i) solutions requiring some kind of support from the network layer (i.e., quality-of-service-based mechanisms);

(ii) solutions based on mechanisms totally independent from the network layer's guarantees and assumptions.

Following approach (i) in [11], a quality-of-service (QoS) extension has been proposed to mobile ad hoc routing in order to support real-time applications. In this case, the main achievement was the reduction of the loss rate, while maintaining acceptable latency and jitter. A radically different approach (ii) is followed by [7] proposing a framework, called Rendezvous, based on an optimistic synchronization scheme that provides a consistency mechanism for high-latency environments. A more general approach can be found in [8]: in this work different mechanisms are introduced and analyzed, to deal with inconsistencies due to the distributed nature of the gaming architecture. In our opinion, at least in a foreseeable future, the majority of wireless networks will not provide any form of QoS to real-time applications. The PS mechanism falls within the approach previously defined as (ii). Upon that premise, it is worth noting that a QoS-enabled network layer would be complementary to the proposed mechanism.

4. ON DESIGNING A PLAYER SESSION

We define playability as the user's satisfaction while playing. In other words, this means that the gameplay quality is related to the "fun to play" and the "usability," with particular attention to the responsiveness and the sensation

of a realistic participation. Moreover, the playability is also related to other typical aspects of a game, such as: storyline quality, customizability, control, intricacy, strategy, and the realism's degree. In our case, we are only interested in the responsiveness and immersive sensations because they are directly related to the communication performance. On the contrary, other aspects (e.g., the storyline quality) can be considered as features of both the game and its "mechanics."

In this scope, we designed and implemented a new mechanism called playing session (PS) that is in charge of monitoring the client-server communication channel. The mechanism will be triggered by network failures, mainly due to the gamers' movement. The core of the PS mechanism is a new level in the communication stack (see Figure 1). More in detail, the PS is composed by two parts: the participatory framework (PF), in charge of detecting and reacting to the network failures and the mimicking mechanism (MM) that tries to reproduce, with an adequate level of mimesis, the user behavior. In our architecture, the PF is present on both sides (client and server). On the mobile device (i.e., the client in our gaming architecture), the PF is able to detect if the currently used game server is unreachable or reachable with a certain delay. On the game server, the PF is able to detect whether the application on the mobile device is experiencing problems (i.e., the movement of the player has caused a network failure). In this case, the PF gives the control of the player's avatar to the mimicking mechanism (MM) that will be in charge of playing, until the communication channel is restored. In this way, the game can continue the progress avoiding that the whole system is affected by the fault of a single player. On the other side, it is worth noting that a disconnected gamer will not be able to continue to play: he will not be able to play until the connection is restored again.

4.1. Interactivity

The gamers expect to play fluently, without taking care of problems deriving from devices' limitations (e.g., size of the control keys and screen, battery duration) or due to the unreliability of communications [5]. We believe that players should be able to maintain their gaming style independently from the network failures. For example, a stronger player should never loose with a weaker one only because a network failure reduces his ability to interact with the VW. Our idea is to relax the temporal constraints of the game progress up to the sensorial perceptivity threshold. This means that the length of a single turn is upper bounded by this threshold. In this way, it is possible to use all the available time to wait for "delayed moves" (i.e., events delayed due to network failures). We should keep in mind that losing an action, with low frequency, is not so critical neither for the game system nor for the player. In case of missing moves, efficient predictive techniques exist (e.g., dead reckoning mechanisms) [12]. Some games are so "fast and furious" that players have not a detailed perception of the whole situation. In this context, it is possible to adaptively decrease or increase the duration of each turn. In detail, the adaptation mechanism could be based on the users' responsiveness.

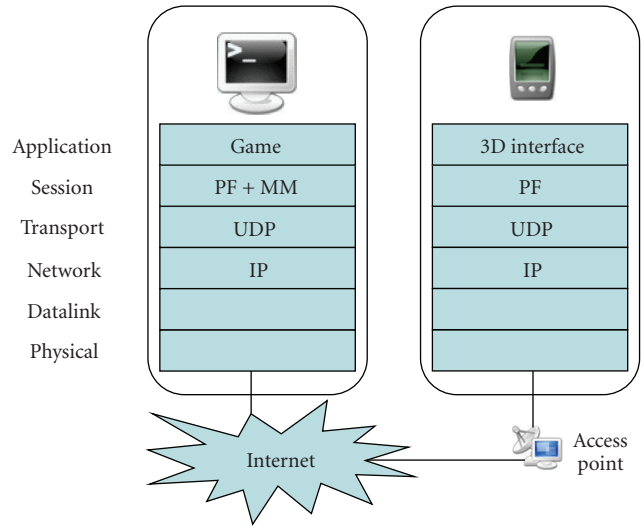


Figure 1: The playing session: a new layer in the communication protocol stack.

In few words, it is mandatory to maintain the temporal order of the moves: it would be possible to decrease or increase their frequency, but always under the sensorial perceptivity threshold. In particular, from the participants' point of view, we do not accept as a solution (aimed at enhancing the playability) the presence of ghost avatars in the scene. In the gaming jargon, the word "ghost" refers to a "frozen" avatar that is not reachable by its gamer.

4.2. Coherence

It is really unpleasant, for gamers, to find frozen avatars in the middle of the scene. A common and really unsatisfactory solution to this problem is to set some prefixed actions to support an "orphan avatar" (i.e., an avatar that is disconnected from its gamer). In order to avoid this unpleasant experience, it is important to maintain a good level of coherence inside the game. In this case, we define coherence as uniformity among players: each player should be able to participate in the game in the most correct way, without limitations severely degrading his experience. In a perfect world, the network communications are reliable and effective, the input device has a good level of usability, the operating system does not crash, and the game system does not fail. The real world is very different: network faults are frequent and the usability of the input devices is often unsatisfactory. In this scenario, an MM would be able to increase the level of the game coherence: for example, it could replace the player should his actions not arrive to the game server within the perceptivity threshold. An important assumption is that the mechanism should be able to play at the same level of the substituted gamer. An MM that plays better than the gamer would make pointless his future actions or introduce a new form of cheating. Conversely, an MM worse than the gamer could reduce his chance of victory, generating inadequate actions that are very difficult or impossible to recover by the gamer. An interesting

side effect is that the other players should not be able to detect which avatars are controlled by a real gamer and which are controlled by the MM. From a different point of view, it should be impossible to detect which gamers are experiencing network problems by simply looking at the behavior of their avatars. In this connection, the MM would be completely transparent to other players. The research and implementation of this mechanism is out of the scope of our work because is mainly related to artificial intelligence (details can be found in [13]), but some considerations useful to improve the PS effectiveness will be following discussed.

Unfortunately, the believability of an avatar is subjective, since it is influenced by the culture and the skills of the other players [14]. Moreover, in order to show an adequate degree of humanness [15], the avatar should adopt human-like reaction and decision times, avoid to give superhuman capabilities and realize some tactical/strategy reasoning [16]. After these considerations: what is the best action that the MM should play when it is in control of an orphan avatar? In this case, the main point is to define which is the exact meaning of "best action" [13]. There are, at least, two different viewpoints:

(i) the player's viewpoint;

(ii) the viewpoint of the other gamers.

In the former case, the best action would be the most predictable one; in the latter, it would be the action that is able to reproduce the strategy of the user. Furthermore, in (i), the new action should be the natural consequence of the correct progress of the game, and in this case, it could be quite easy to be predicted. This approach is effective only if the gamer loses the control of the avatar for a very limited amount of time, and with low frequency. The main advantage is related to the implementation: it could be based on a simple lookup table of state-action pairs. An interesting example of a related technique is dead reckoning [12]: in this case the prediction of the future state of the avatar is based on the current state (e.g., the future position of the avatar is forecast taking in account its current position, speed, and direction). Unfortunately, a mechanism based on hard-coded default actions is unable to comply with long-term disconnections: the avatar would be quickly recognized as a fake. If most part of the actions are played by the MM or if it is triggered too often, then the avatar will likely start to show nonhuman behaviors. In this case, in order to maintain an acceptable level of coherence, the MM should also take into account other factors, as an example the stochastic/strategic behavior of real gamers. Traditionally, this problem has been solved by increasing the complexity of the algorithms used to control the avatar. These algorithms are not easy to be designed and implemented because many combinations of events and situations have to be considered and some of them are very hard to be predicted in advance. Following this approach, it would be possible to extend some of the dead reckoning concepts. For example, an extended dead reckoning for first person shooters (FPSs) would require supporting many actions such as: jumping, changing the weapon, shooting enemies, and in some cases also more complex actions (e.g., setting a trap). An alternative approach is to raise the level of abstraction to a tactical or strategic level [17]. In practice, it would be possible to monitor each user to infer his typical gaming behavior: the MM would be instructed to follow the strategy of the gamer. Furthermore, the MM should be adaptive and able to capture the real essence of the strategy, instead of a collection of disconnected actions. In this sense, we need techniques [18] capable of analyzing a collection of task pairs (instance, solution) without knowing the dynamics of the solution (i.e., without formalizing the algorithm). With a collection formed by an adequate number of instances, an MM should be able to substitute the player's strategy/ability with an appropriate level of mimesis and with a good level of generalization. Another problem is related to the computational effort required to obtain timely results. The MM should be able to infer a "good action" in a short time: also in this case the amount of available time is bounded by the perceptivity threshold. Finally, different approaches can be followed in the production of the model knowledge. A first approach would be to collect offline the data for instructing the MM. On the other side, a more complex and costly approach would collect data during the game progress (i.e., online). In this case, it would be possible to dynamically adapt the MM mechanism to the strategy evolution and to different gaming events.

4.3. Equity/fairness

Interactivity and coherence are the bases for achieving a good level of equity/fairness in the game. In a distributed system, fairness can be defined as the guarantee to avoid the starving of any process: each process should have the same priority in the access of shared resources. In this case, all processes should have the same chance to progress. In gaming, the aim is to guarantee substantial equity among all players. In a perfect world, each gamer would be allowed to play the same number of actions, with the same frequency of the other participants. Unfortunately, problems due to network communications can have a high impact on the game equity. From our point of view, avatars and gamers should be decoupled, we consider avatars as processes that, in case of network failures, are separate from gamers.

The chance for each gamer to play the same number of actions with the same frequency in the match is a key point to evaluate the game equity, and therefore both aspects should be carefully measured. As a consequence, if the PF is able to promptly detect the network failures and in case of missing events, to substitute the gamer, then it would be possible to ensure fair gaming conditions. It is worth noting that this does not mean that all gamers, at the end of the match, will have played the same exact number of actions. Some gamers could have played fewer actions with respect to others, but in any case, the number of processed actions will be the same for each avatar. In this sense, the PS mechanism gives all gamers the same chance to win the game.

5. SYSTEM ARCHITECTURE

The PS mechanism has been implemented using a Multiagent system (MAS). In detail, the prototype has been integrated in the system for parallel agent discrete event simulation (SPADES) [19], a well-known MAS. On this basis, we added a PS layer in the protocol stack (as shown in Figure 1). As said in Section 4, the PS is in charge of monitoring the communication between the gamer and his avatar and to react in presence of failures or delays. In particular, our system architecture must be able to cope with two different situations:

(i) the loss of a low percentage of actions, with a low frequency (i.e., short interruptions);

(ii) the loss of a substantial percentage of actions (i.e., a train of actions) or low percentage with high frequency (i.e., long or permanent interruptions).

In the first case (i), the PF detects the problem and tries to maintain a good level of interactivity: forcing MM to control the orphaned avatar and hence to produce moves within the perceptivity threshold and with an adequate level of mimesis. In the latter (ii), the PF will try to resume the control of the gamer on the avatar and, in the meantime, MM will produce an adequate strategy to control the avatar. As a consequence, PF (see Figure 2) is composed by a couple of modules, the user participatory framework (UPF) and the avatar participatory framework (APF). The UPF, which is accommodated in the gamer device, checks the state of communications and verifies if the gaming architecture is reachable or not. On the server side, APF monitors the communications with gaming devices.

5.1. Avatar participatory framework (APF)

At the beginning of a match, the APF initializes a UDP communication to the UDF. In the meantime, the MM will control the avatar until it starts receiving actions from the gamer. The communication channel is used by the gamer to take the ownership of a specific avatar: usually, the pairing between the gamer and his avatar will last for the whole game duration. The APF tries to maintain active the communication with the corresponding UPF, in case of a long or permanent interruption, it will wait for the recovery of the avatar. When the communication is active, the APF continuously checks two different timeouts. The former (i.e., the action timeout) prevents the slowdown of the game progress due high latency in the interactions between the gamer device and the server. This is implemented by the APF monitoring the responsiveness of the related UPF: the measured latency has to remain under a predefined upper bound (i.e., the perceptivity threshold). Clearly, as above mentioned, if the upper bound is exceeded then the APF forces the MM to play an action in place of his gamer. If the MM plays a number of consecutive times that exceeds a maximum value (defined as transport timeout), then the APF sets the state of the communication as "broken." As a direct consequence, the APF shuts down the existing communication channel and changes its state to "listening" mode, and waits for the recovery of the avatar.

5.2. User's participatory framework (UPF)

The UPF (placed in the gaming device) continuously checks whether its avatar is reachable, or if the recovery of the communication is necessary. As described in Section 5.1, the UPF takes the ownership of a specific avatar establishing a communication channel. Every turn, the protocol forces each APF to send a RequestAction event to its UPF. The UPF waits a RequestAction event for a period that is no longer than an action timeout. Furthermore, each RequestAction event is identified by an incremental number. In this way, it is possible to detect if RequestAction events were delayed or lost (i.e., a network failure has occurred). If the RequestAction event comes too late, the UPF buffers the last user-generated action, waiting for the next timeslot. In the meantime, if the player generates another action, then the UPF will overwrite the previously buffered one. In this way, it avoids the delivery of an action that is related to an old state of the game. If the UPF does not receive any event within the transport timeout, then it sets the state of the communication as "broken." In this case, the transport timeout is equal to the maximum number of consecutive action timeouts that can be exceeded: it is worth noting that this value depends on the specific semantics of each game. Finally, if the communication state is set as broken, then the UPF shuts down the existing communication channel and tries to restore the control of its avatar, instancing a new channel.

6. CASE STUDY

In this section, we claim that, even in presence of network failures, the PS mechanism maintains the interactivity within the sensorial perceptivity threshold and does not alter the gamers' strategy. As a consequence, the PS will not alter the progress of the virtual world making the chances of victory of each player unaltered. The PS is completely transparent to the gamer: it is not invasive, it does not affect the satisfaction of the gamer and enhances equity and playability. Its performance measures are strictly related to the following conditions:

(i) C1: the game engine progress respects the sensorial perceptivity threshold;

(ii) C2: each avatar plays the same number of actions, with the same frequency;

(iii) C3: the PS mechanism does not alter the chances to win of each gamer.

If all conditions are met then the PS mechanism is able to maintain a good level of interactivity and coherence, guaranteeing equity between gamers. To support this thesis, a clone of the Armagetron game [20] (inspired by the light-cycles sequence in the Disney movie Tron) has been implemented on top of the PS prototypal implementation. Armagetron is a multiplayer game where participants challenge each other driving a "synthetic motorbike" that leaves behind a wall.

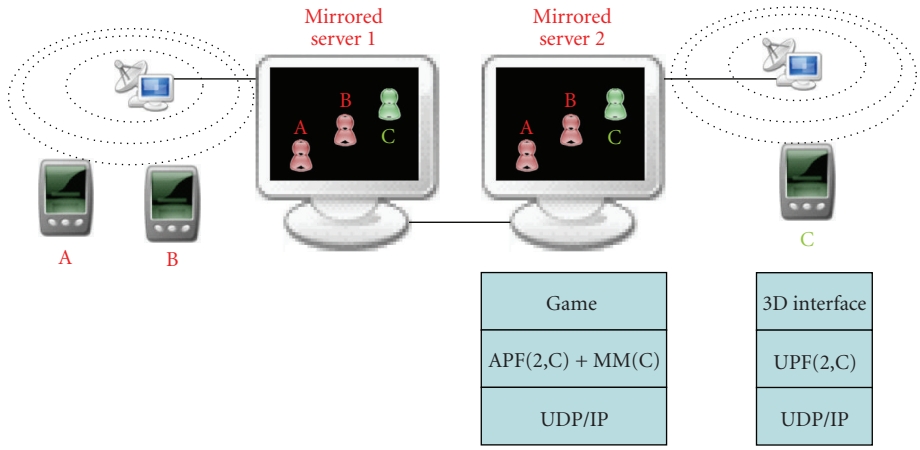


Figure 2: The playing session mechanism.

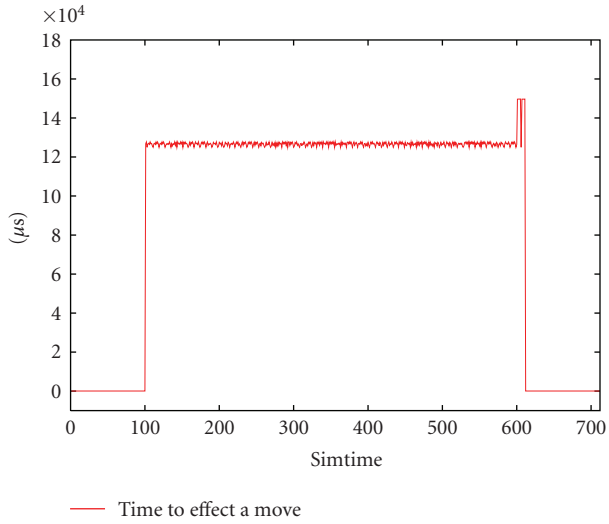


Figure 3: Timeline of the game progress with L = 125 milliseconds and PL = 0%.

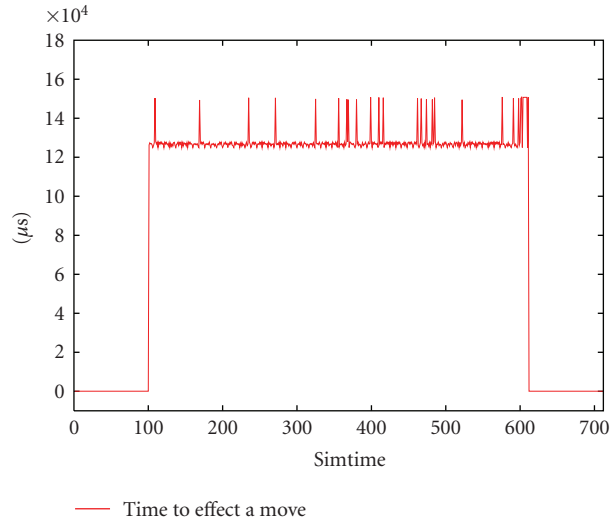


Figure 4: Timeline of the game progress with L = 125 milliseconds and PL = 5%.

During the drive, the motorbikes have to avoid the walls: if a motorbike crashes into a wall or into the borders of the arena then game is over. The aim of the game is to stay alive while killing other player, blocking their path. The winner is the last one alive. To make the game more complex, a motorbike can never stop: it can only accelerate up to the maximum speed and decelerate to the minimum. Turning left or right slows down the speed of the motorbike. Armagetron is interesting because it is a fast-paced multiplayer game, it has a low-complexity implementation but supports sophisticated strategies.

6.1. Network performance

In order to study the performance and the effectiveness of the PS mechanism, we emulated [21] five different network scenarios with increasing packet loss (PL) ratio (0–20%). In this case, the latency (L) was set to 125 milliseconds. We repeated the same matches under different scenarios, in order to study the invasiveness of the proposed mechanisms. In our opinion, the variation of the PL ratio can be used to reproduce the typical network problems of a gamer wandering about the city. The different PL rate should be able to reproduce the following situations:

(i) the gamer is near to the AP and, therefore, the signal strength is very good (i.e., 0% PL);

(ii) the signal is attenuated by obstacles (i.e., 5% and 10% PL);

(iii) the gamer is moving in and out of the coverage area (i.e., 20% PL).

Table 1 reports the average (μ) and the standard deviation (σ) (in microseconds) of the time required to process a turn, the duration of the match (Σ, in microseconds), the percentage of actions played by each gamer with respect to

Table 1: Performance evaluation of PS with different packet losses.

PL	μ (μs)	σ (μs)	Σ (μs)	% played actions (gamer)	# actions (gamer + MM)
0%	126994	3313	64893734	100	500
5%	127950	5633	65382227	96	500
10%	128736	6861	65784246	92.4	500
20%	131010	9289	66946053	82.4	500

Table 2: Evaluation of different strategies.

Strategies	Average occupied space (%)	σ	Average lifetime (simtime)	σ
(I)	6.4	0.8	145	35.9
(II)	6.9	0.7	127	32.1
(III)	4.3	0.4	442	133.1
(IV)	6.5	0.8	561	133.6

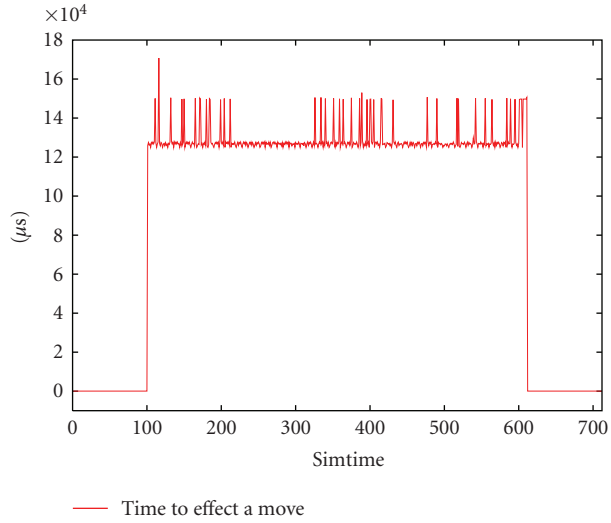


Figure 5: Timeline of the game progress with L = 125 milliseconds and PL = 10%.

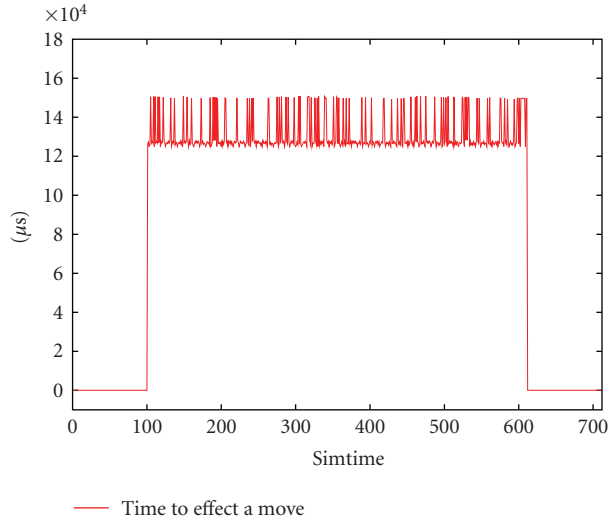


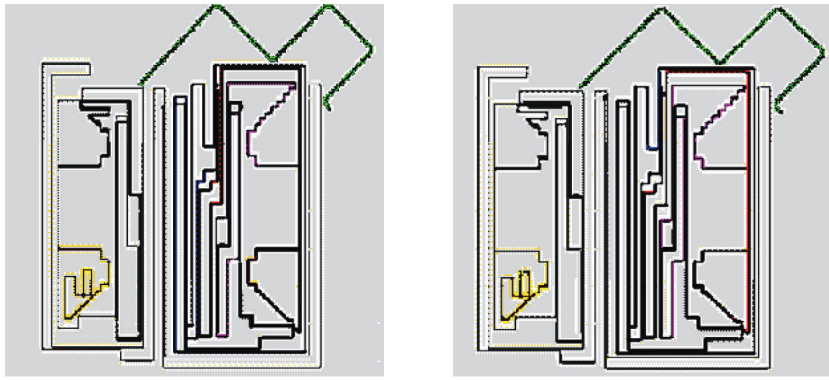
Figure 6: Timeline of the game progress with L = 125 milliseconds and PL = 20%.

the total number of actions played by its avatar, and finally the number of actions played by the related avatar (i.e., the sum of the actions played by the gamer and his MM). Figures 3–6 show a timeline of the game progress: the X-axis represents the simulated time, expressed in turns, (simtime), the Y-axis represents the wall-clock-time required to process a turn (in microseconds). 500 actions are represented in all figures, the game starts at simtime 100 and goes on until the last action is executed. Figures show that the time required for a turn is always under the sensorial perceptivity threshold (150 milliseconds), even in critical situations (i.e., high levels of packet loss). In this sense, the condition C1 is satisfied. Furthermore, the last column in Table 1 shows that the total number of played actions does not depend on the scenario. In detail, the frequency of played actions is comparable, and this is demonstrated by μ, σ, and Σ (see second, third, and fourth columns of the table). In this sense, also the condition C2 is verified.

6.2. Effectiveness of different gaming strategies

In order to show that the PS mechanism is not "invasive" (i.e., does not alter gaming outcomes), we have verified how much the results of the same match played in different network scenarios diverge. In our opinion, the chances of victory of a strategy, with respect to another one, should remain unchanged, despite the activation of the PS mechanism. In order to produce an adequate number of trials for the comparisons, we have automated the gaming process, reproducing the most common strategies used in Armagetron. In detail, we created a mechanism (called gamer equivalent (GE)) used to simulate the behavior of gamers with different levels of ability (i.e., from newbie to expert). The GE is implemented in the UPF module and supports four strategies:

(I) the first strategy mimics the behavior of a newbie. If the avatar is crashing into a wall then it can turn left or right. It will check both directions and choose the one without obstacles;



(a) (b)

Figure 7: Snapshots of two runs of the same match: (a) unreliable communications, (b) reliable communications.

(II) the second strategy tries to find the direction with the longest free path up to a wall;

(III) the third strategy chooses the direction with the highest number of available paths in the next turn;

(IV) the fourth strategy is very similar to the previous one but it attempts at forecasting the state of next turns instead of just one.

The following performance evaluation is based on 30 trials; in each trial all gamers follow the same strategy. The main results collected are the amount of space occupied by the wakes (generated by motorbikes) and the mean lifetime (see Table 2). For the sake of clearness, in Table 3 only the results obtained by the best strategies are reported. By comparing Tables 2 and 3, we see that the lifetime parameter is more important than the occupied space. For example, strategy (III) is successful versus (I) and (II), even if (I) and (II) have covered a greater percentage of the arena with their wakes. Strategy (III) seems better in exploiting the space near the wakes generated by other gamers.

As described in Section 5, the PS introduces a quite complex timeouts' management that leads to a different timing of actions with respect to a standard game execution. The goal of the last part of the evaluation is to demonstrate that the PS mechanism does not significantly alter the game progress. In this test-bed evaluation, this is done by using in each MM the same gamer equivalent (GE) that has been implemented in the UPF. Mainly, if an action generated by the GE in the UPF does not reach the avatar, then the GE inside the MM will produce exactly the same action. This has been done in order to eliminate any interference due to the MM mechanism. Table 3 shows the results in presence of a reliable and unreliable communication. In the former case, the PL was set to 0%, in the latter, the PL was from 5% to 20%. These results show that, in presence of the PS mechanism, the unreliability of communication does not affect the general outcome of the strategies. The small differences that can be found in Table 3, are due to the random components in the implementation of the GE. To verify this hypothesis, tests have been repeated without the random generators. Figure 7 reports two snapshots of the same match: the first one (left side) is obtained in presence of unreliable communications (20% of PL), the latter (right side) with reliable communications. It is easy to see that the progress of the match and the wakes of the avatars are exactly the same (the very small differences in the visualization are due to the graphic engine). After this result, we can conclude that the performance of a strategy is not altered by the MM, even if the gamer is frequently substituted. Therefore, the PS mechanism has not altered the chances to win of each gamer, and, in this sense, also the C3 condition is verified.

Table 3: Deathmatch of strategies: matches won (%) with reliable and unreliable communications.

Match	Unreliable communications	Reliable communications
(I) versus (III)	0–100	0–100
(II) versus (III)	4–96	0–100
(III) versus (IV)	0–100	8–92

7. CONCLUSIONS AND FUTURE WORK

Virtual environments are an implementation of the metaverse concept, a simulated world populated by a massive number of synthetic avatars that are controlled via Internet. The telecommunication industry is fostering the "ubiquitous participation" of users in virtual environments, producing and marketing powerful mobile devices with wireless capabilities. In this sense, the support of nomadic users in massive metaverses is a very hot topic in research and business. The communication unreliability is one of the main characteristics of wireless technologies and mobile environments. This aspect is very important when dealing with virtual environments, since it can significantly reduce the effectiveness of the distributed architecture and severely limit the playability of the game. In this paper, we propose a new mechanism (called playing session (PS)) that aims at solving this problem, by introducing an architecture capable of dealing with network failures. In order to evaluate our proposal, we implemented a clone of the Amagetron game

based on a prototypal implementation of the PS mechanism. The performance evaluation of the case study has shown that the mechanism enhances the playability of the game, while assuring a good level of equity among users.

Future works should extend the prototypal implementation, investigate the subjective expectations of gamers, and consider the cheating problem. From a technical viewpoint, the current implementation of the PS mechanism works in the client-to-server side of the gaming architecture. As a future evolution, the mechanism could be also extended to the server-to-server side: in this case, aiming at reducing the impact of network failures in the communications among servers.

ACKNOWLEDGMENT

Authors would like to thank the anonymous referees for their contributions aimed at improving the paper quality.

REFERENCES

[1] World of Warcraft, 2007, http://www.worldofwarcraft.com/index.xml.

[2] Second Life, Linden Research, Inc., 2005, http://secondlife.com/.

[3] Project Entropia, 2006, http://www.entropiauniverse.com/en/rich/5000.html.

[4] Sociolotron, 2006, http://sociolotron.amerabyte.com/website-2/intro.htm.

[5] S. Cacciaguerra, S. Mirri, M. Roffilli, and P. Salomoni, "Let me participate! Using intelligent agents to support inclusive playing for gamers in disadvantaged conditions," *WSEAS Transactions on Communications*, vol. 5, no. 10, pp. 1973–1980, 2006.

[6] M. Furini, "Mobile games: what to expect in the near future," in *Proceedings of GAMEON Conference on Simulation and AI in Computer Games*, Bologna, Italy, November 2007.

[7] A. Chandler and J. Finney, "On the effects of loose causal consistency in mobile multiplayer games," in *Proceedings of the 4th ACM SIGCOMM Workshop on Network and System Support for Games*, pp. 1–11, Hawthorne, NY, USA, October 2005.

[8] J. Brun, F. Safaei, and P. Boustead, "Managing latency and fairness in networked games," *Communications of the ACM*, vol. 49, no. 11, pp. 46–51, 2006.

[9] T. Beigbeder, R. Coughlan, C. Lusher, J. Plunkett, E. Agu, and M. Claypool, "The effects of loss and latency on user performance in unreal tournament 2003," in *Proceedings of the 3rd ACM SIGCOMM Workshop on Network and System Support for Games*, pp. 144–151, Portland, Ore, USA, August 2004.

[10] N. Sheldon, E. Girard, S. Borg, M. Claypool, and E. Agu, "The effect of latency on user performance in warcraft III," in *Proceedings of the 2nd Workshop on Network and System Support for Games (NetGames '03)*, pp. 3–14, Redwood City, Calif, USA, May 2003.

[11] K. Farkas, D. Budke, B. Plattner, O. Wellnitz, and L. Wolf, "QoS extensions to mobile ad hoc routing supporting real-time applications," in *Proceedings of the IEEE International Conference on Computer Systems and Applications (AICCSA '06)*, pp. 54–61, Dubai, UAE, March 2006.

[12] L. Pantel and L. Wolf, "On the suitability of dead reckoning schemes for games," in *Proceedings of the 1st Workshop on Network and System Support for Games (NetGames '02)*, pp. 79–84, Braunschweig, Germany, April 2002.

[13] S. Cacciaguerra and M. Roffilli, "Agent-based participatory simulation activities for the emergence of complex social behaviours," in *Proceedings of the Social Intelligence and Interaction in Animals, Robots and Agents (AISB '05)*, pp. 1–29, Hatfield, England, April 2005.

[14] B. Mac Namee, *Proactive persistent agents: using situational intelligence to create support characters in character-centric computer games*, Ph.D. dissertation, University of Dublin, Dublin, Ireland, 2004.

[15] D. Livingstone, "Turing's test and believable AI in games," *Computers in Entertainment*, vol. 4, no. 1, article 6, pp. 1–13, 2006.

[16] S. Mc Glinchey and D. Livingstone, "What believability testing can tell us," in *Proceedings of the Conference on Game AI, Design and Education (CGAIDE '04)*, p. 273, Reading, UK, November 2004.

[17] J. Smed and H. Hakonen, "Three concepts for light-weight communication in multiplayer games," in *Proceedings of the 1st International Digital Games Conference (iDiG '06)*, pp. 199–202, Portalegre, Portugal, September 2006.

[18] T. G. Dietterich, "Machine learning research: four current directions," *AI Magazine*, vol. 18, no. 4, pp. 97–136, 1997.

[19] P. Riley, "SPADES: a system for parallel-agent, discrete-event simulation," *AI Magazine*, vol. 24, no. 2, pp. 41–42, 2003.

[20] Armagetron, 2005, http://armagetronad.net/.

[21] S. Cacciaguerra, *Experiences with synthetic network emulation for complex IP based networks*, Ph.D. dissertation, University of Bologna, Bologna, Italy, 2005.

Hindawi Publishing Corporation
International Journal of Computer Games Technology
Volume 2008, Article ID 906931, 9 pages
doi:10.1155/2008/906931

Research Article

Visualization of Online-Game Players Based on Their Action Behaviors

Ruck Thawonmas[1] and Keita Iizuka[1, 2]

[1] *Intelligent Computer Entertainment Laboratory, Graduate School of Science and Engineering, Ritsumeikan University, Kusatsu, Shiga 525-8577, Japan*
[2] *Solution Development Team, Solution Development Department, Bandai Networks Co., Ltd., Tokyo 111-8081, Japan*

Correspondence should be addressed to Ruck Thawonmas, ruck@ci.ritsumei.ac.jp

Received 1 February 2008; Revised 8 May 2008; Accepted 11 June 2008

Recommended by Jouni Smed

We propose a visualization approach for analyzing players' action behaviors. The proposed approach consists of two visualization techniques: classical multidimensional scaling (CMDS) and KeyGraph. CMDS is for discovering clusters of players who behave similarly. KeyGraph is for interpreting action behaviors of players in a cluster of interest. In order to reduce the dimension of matrices used in computation of the CMDS input, we exploit a time-series reduction technique recently proposed by us. Our visualization approach is evaluated using log of an online game where three-player types according to Bartle's taxonomy are found, that is, achievers, explorers, and socializers.

1. INTRODUCTION

The market size of online games continues to experience surging growth [1]. At the same time, competitions among them are also becoming very high. The quality of player service plays an important role in winning such competitions. It is therefore inevitable to online-game developers and publishers to know their player behaviors so that they can develop game contents that fulfill player demands. Visualization techniques have been recently applied to discover in-game player behaviors.

Most work in the literature focuses on visualization of player trails or time series of visited locations for examining the distance over time among the members of a social group [2], discovering playing strategies in a combat game [3], and analyzing movement patterns [4]. Other work focuses on extracting pathways [5] and on locating clusters of similar players based on their movement patterns [6].

This research, however, focuses on visualizing player behaviors based on their actions. According to Bartle's taxonomy [7], online-game players can be typically identified based on their action behaviors into achievers, explorers, killers, and socializers. Player-type information should, therefore, be exploited to provide game contents that players favor, for example, a wider variety of collectable items for achievers, longer missions for explorers, more hunting opportunities for killers, and a higher frequency of social events for socializers. It has been recently reported in [8] that Bartle's taxonomy is also applicable to social data in a web-based application.

In this paper, we propose an approach for visualizing players' action behaviors using classical multidimensional scaling (CMDS) [9] and KeyGraph [10], both described in Section 2. First, CMDS is used for locating clusters of similarly behaving players. KeyGraph is then used for interpreting playing behaviors of players in a cluster of interest. The input to CMDS is derived based on time-series matrices of players' action sequences which are needlessly long due to noise and redundancy, leading to high computational cost. We, therefore, compute the CMDS input based on reduced time-series matrices obtained by our recently proposed time-series reduction technique in [11]. To make this paper self-contained, this technique is described in Section 3. Evaluation of our visualization approach is

given in Section 4, where achievers, explorers, and socializers are found in play log from an online-game used in the evaluation.

2. PLAYER VISUALIZATION

In this section, we describe CMDS, KeyGraph, log format, and visualization metrics. As with most other tools for information visualization [12], subjective interpretation is required for KeyGraph. The described visualization metrics are used for facilitating this task in Section 4.2.

2.1. Multidimensional scaling

CMDS is a prevailing technique for mapping pair-wise relationships to coordinates and has been applied to several areas such as statistics, psychology, sociology, political sciences, and marketing [9]. Recently, this technique has been successfully applied to clustering of online-game players based on their movement patterns [6]. CMDS takes as its input matrix **D**, indicating dissimilarities between player pairs, and outputs a coordinate matrix whose configuration minimizes a loss function in preserving all interpoint distances. Two time series of interest are considered similar if they have similar rise and fall patterns, although they might have different scales on the time axis. A good measurement for deriving the distance or dissimilarities between such series is the dynamic time warping (DTW) distance [13].

In our research, the ijth element in **D** is the DTW distance between the reduced time-series matrices of action sequences of players i and j. In addition, we use the function cmdscale in the Statistical Toolbox of Matlab for performing CMDS and select only the first two dimensions of the constructed coordinates for plotting players.

2.1.1. Action coding

Here, we describe how action sequences are numerically coded into time-series matrices for computation of DTW distances. Let O denote the set of action symbols of interest and $|O|$ its cardinality. Action sequence $s = s(1), s(2), \ldots, s(L)$ is numerically coded into $|O| \times L$ time-series matrix $\mathbf{X} = [X(1), X(2), \ldots, X(L)]$, where $X(i)$ is a column vector with the element indexing the action symbol of $s(i)$ being 1 and other elements 0.

Consider, for example, the set of action symbols $O = \{\mathrm{A}, \mathrm{B}, \mathrm{C}\}$, and thus $|O| = 3$, where symbols A, B, and C are represented by column vectors $[100]^t$, $[010]^t$, and $[001]^t$. In an action sequence such as $s = \mathrm{C}, \mathrm{A}, \mathrm{B}, \mathrm{C}$, it is coded into $\mathbf{X} = [[001]^t, [100]^t, [010]^t, [001]^t,]$.

2.1.2. Dynamic time warping

The DTW distance between time-series matrices **X** and **Y**, $\mathrm{dtw}(\mathbf{X}, \mathbf{Y})$, having lengths $L_{\mathbf{X}}$ and $L_{\mathbf{Y}}$, is defined as follows:

$$\mathrm{dtw}(\mathbf{X}, \mathbf{Y}) = g(L_{\mathbf{X}}, L_{\mathbf{Y}}), \tag{1}$$

A	C	B	E	F
1	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	0	0	0	0
0	0	0	1	0
0	0	0	0	1

A	B	C	E	D	F	B
1	0	0	0	0	0	0
0	1	0	0	0	0	1
0	0	1	0	0	0	0
0	0	0	0	1	0	0
0	0	0	1	0	0	0
0	0	0	0	0	1	0

Figure 1: Time-series matrices **X** and **Y**.

F	5.6	4.2	4.2	4.2	4.2	4.2	5.6
E	4.2	2.8	2.8	2.8	4.2	5.6	7
B	2.8	1.4	2.8	2.8	4.2	5.6	5.6
C	1.4	1.4	1.4	2.8	4.2	5.6	7
A	0	1.4	2.8	4.2	5.6	7	8.4
	A	B	C	E	D	F	B

Figure 2: Derivation of dynamic time warping distance between **X** and **Y**.

where

$$g(i,j) = \min \begin{cases} g(i, j-1) + d(i,j), \\ g(i-1, j-1) + d(i,j), \\ g(i-1, j) + d(i,j), \end{cases}$$

$$g(i,0) = \begin{cases} 0 & i = 0, \\ \infty & i > 0, \end{cases} \tag{2}$$

$$g(0,j) = \begin{cases} 0 & j = 0, \\ \infty & j > 0, \end{cases}$$

and $d(i,j)$ is the Euclidean distance between $X(i)$ and $Y(j)$.

Consider, for example, the set of symbols $O = \{\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}, \mathrm{E}, \mathrm{F}\}$ and two action sequences $x = \mathrm{A}, \mathrm{C}, \mathrm{B}, \mathrm{E}, \mathrm{F}$ and $y = \mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{E}, \mathrm{D}, \mathrm{F}, \mathrm{B}$. The DTW distance between corresponding time-series matrices **X** and **Y** (c.f., Figure 1), $\mathrm{dtw}(\mathbf{X}, \mathbf{Y})$, is 5.6, derived as shown in Figure 2.

2.2. KeyGraph

KeyGraph is a visualization tool for discovery of relations among text-based data. Its underlying concept is based on a building construction metaphor. As shown in [10], the precision-recall curve of KeyGraph is superior to TFIDF and NGRAM [14], well-known techniques for information retrieval, in extraction of correct keywords from a set of documents. KeyGraph has been later applied to visualize the relations among Web pages, among products in markets, among earthquake faults, and so forth [15]. It has also been successfully applied to identification of player types in an online-game simulator [16].

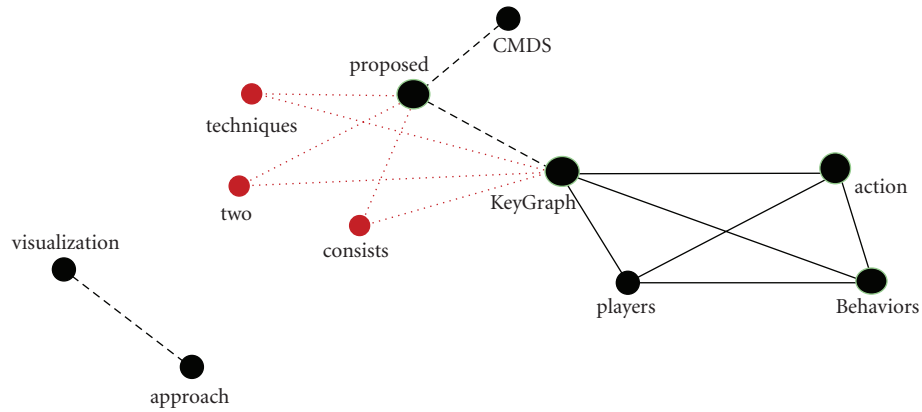


Figure 3: KeyGraph applied to the abstract of this paper.

Three major components of KeyGraph are as follows.

(1) Foundations: subgraphs of highly associated and frequent terms representing basic concepts in the data.

(2) Roofs: terms highly associated with foundations.

(3) Columns: associations between foundations and roofs used for extracting keywords or main concepts in the data.

Associations between terms are defined as the co-occurrence among them in same sentences, and keywords are the terms in either foundations or roofs that are connected to strong columns. Under KeyGraph representation, solid lines and their touching black nodes depict foundations, dotted lines depict columns, red nodes depict roofs excluding those in the foundations, and double circles depict keywords. We use a tool called Polaris [17], publicly available, for generating KeyGraphs (http://www.chokkan.org/software/dist/polaris-0.19alpha.zip).

Figure 3 shows an example of KeyGraph when it is applied to the text data taken from the abstract of this paper, where common preprocessing for text data such as removing of conjunctions, determiners, and prepositions is performed. From this figure, it can be seen that there is one foundation consisting of four terms, that is, "KeyGraph," "action," "behaviors," and "players." The first three terms are also keywords. Another keyword is "proposed." Three roof terms are "consists," "two," and "techniques." These terms well represent the messages in the abstract.

2.3. Log format and visualization metrics

Player action sequences in our work are sequences of action symbols extracted from game log, of an online game discussed in Section 4, that has the following format:

$$\text{time stamp}\|\text{player ID}\|\text{event}\|\text{start position}\|\text{stop position}\|, \quad (3)$$

where an event consists of an action and its object, if any. This format is based on those adopted in an MMOG simulator in [18] and 3D virtual worlds in [2]. In commercial online games, this kind of game log is stored in the monitoring database [19].

According to a recent work in [20], there are three categories of play motivations in online games as follows.

(i) Achievement consisting of three subcomponents, that is, advancement, mechanics, and competition.

(ii) Social consisting of three subcomponents, that is, socializing, relationship, and teamwork.

(iii) Immersion consisting of four subcomponents, that is, discovery, role-playing, customization, and escapism.

The achievement, social, and immersion categories correspond to Bartle's achievers, socializers, and explorers, respectively, although the above ten motivations overlap among player types.

In our work, we focus in particular on

(i) advancement described in [20] as the desire to gain power progresses rapidly and accumulates in-game symbols of wealth or status,

(ii) socializing described in [20] as having an interest in helping and chatting with other players, and

(iii) discovery described in [20] as finding and knowing things that most other players do not know about.

This is because we anticipate that they should be identifiable using our action sequences and KeyGraphs. Below, we verify this anticipation with simplified data sets and their KeyGraphs, which serve as our visualization metrics for facilitating interpretation of KeyGraph results in Section 4.2.

Let us consider a set of action symbols {c, w, m, n, r}, standing for chat, walk, interaction with a mission master, interaction with a nearby object (item, NPC, or monster), and interaction with a remote object, respectively. The symbol w is a fundamental action and thus should be a frequent symbol in all action sequences. It is therefore removed from our consideration. For achievers motivated by advancement, interactions with mission masters should

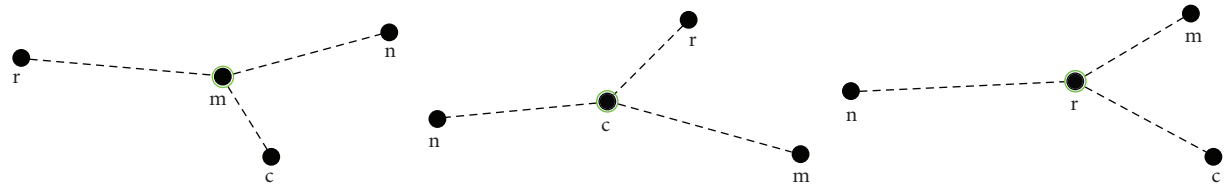


Figure 4: (a) KeyGraphs for achievers, (b) socializers, and (c) explorers generated from the given sample data sets, where m (interaction to a mission master), c (chat), and r (interaction to a remote object) are the keywords in (a), (b), and (c), respectively.

be frequently seen in their action sequences, and thus all possible sets of frequent action symbols for them are {c, m}, {m, n}, {m, r}, {c, m, n}, {c, m, r}, {m, n, r}, and {c, m, n, r}. For socializers motivated by socializing, chats should be commonly seen among them, leading to sets of frequent action symbols {c, m}, {c, n}, {c, r}, {c, m, n}, {c, m, r}, {c, n, r}, and {c, m, n, r}. For explorers motivated by discovery, interactions with remote objects should be commonly seen among them and thus their sets of frequent action symbols should be {c, r}, {m, r}, {n, r}, {c, m, r}, {c, n, r}, {m, n, r}, and {c, m, n, r}.

Figure 4 shows the resulting KeyGraphs for the three player types, where each KeyGraph was generated from the data sets of the corresponding player type. Note that m, c, and r are the keyword nodes in the KeyGraphs of achiever, socializer, and explorer, respectively, and henceforth these findings are used as visualization metrics.

3. TIME-SERIES REDUCTION

Our technique for obtaining compact sequences representing major player behaviors is based on Haar wavelet transform [21]. The Haar wavelet transform technique has a wide range of time-series applications including classification of DNA sequences [22], which motivated us to apply the technique to action sequences. Below, we first give an outline of the Haar wavelet transform and then describe the time-series reduction technique.

In the wavelet transform concept, decomposition involves obtaining wavelet coefficients from a sequence of interest. Reconstruction involves recovering the original sequence from obtained coefficients. Henceforth, it is assumed that the length L of a sequence is adjusted so that L is a power of 2 and $q = \log_2(L)$. The ith Haar wavelet coefficient at resolution order k, $d_{(k,i)}$, is derived as

$$d_{(k,i)} = \frac{x_{(k+1,2i-1)} - x_{(k+1,2i)}}{2}, \tag{4}$$

where $x_{(k,i)} = (x_{(k+1,2i-1)} + x_{(k+1,2i)})/2$ is the ith average at order k between two corresponding adjacent values in order $k+1$. With this representation, $k_{\max} = q$, the original sequence is represented by $x = x_{(q,1)}, x_{(q,2)}, \ldots, x_{(q,L)}$. An example of Haar wavelet decomposition of the sequence 6, 8, 2, 7, 6, 5, 4, 3 is shown in Table 1.

Table 1: Example of Haar wavelet transform.

Resolution	Averages $x_{(k,i)}$	Coefficients $d_{(k,i)}$
$k = 4$	(6, 8, 2, 7, 6, 5, 4, 3)	—
$k = 3$	(7.0, 4.5, 5.5, 3.5)	(−1.0, −2.5, 0.5, 0.5)
$k = 2$	(5.75, 4.5)	(1.25, 1)
$k = 1$	(5.125)	(0.625)

A	B	C	E	D	F	B	A
1	0	0	0	0	0	0	1
0	1	0	0	0	0	1	0
0	0	1	0	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	0	1	0	0

Figure 5: Time-series matrix **X**.

Reconstruction of a given sequence from its Haar wavelet coefficients and averages is done as follows:

$$\begin{aligned} x_{(k,2i-1)} &= x_{(k-1,i)} + d_{(k-1,i)}, \\ x_{(k,2i)} &= x_{(k-1,i)} - d_{(k-1,i)}. \end{aligned} \tag{5}$$

Now, we describe our procedure for reducing the length of the time-series matrix of an action sequence of interest. For explanation, we use action sequence $x = \text{A, B, C, E, D, F, B, A}$ as an example, where $O = \{\text{A, B, C, D, E, F}\}$ and thus $|O| = 6$.

(i) Derive time-series matrix **X** for action sequence x of interest having length L. Figure 5 shows resulting time-series matrix **X** in our example.

(ii) Decompose each row in **X** to obtain Haar wavelet coefficients. Figure 6 shows resulting averages and coefficients for **X** in our example.

(iii) Reconstruct each row in **X** with selected Haar wavelet coefficients as follows.

Reconstruction of each row in **X** starts from the coefficient at the lowest resolution order, that is, $d(1, 1)$, to those at the next higher order, and so forth. At a given resolution order, when the number of remaining coefficients is less than the number of coefficients in that order, they are selected

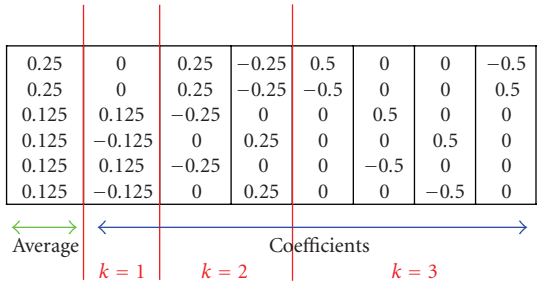


Figure 6: **X** after decomposition.

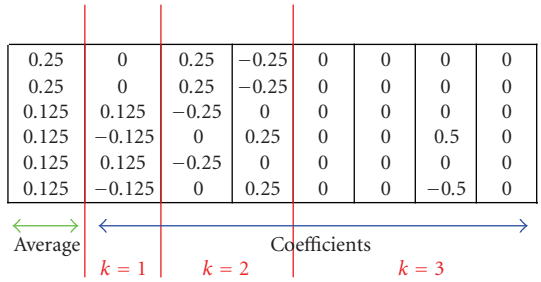


Figure 7: **X** after coefficient selection.

based on their total energy value in decreasing order, where total energy of $d_{(k,i)}$, $E_{(k,i)}$, is defined as

$$E_{(k,i)} = \sum_{n=1}^{|O|} d^2_{(n,k,i)}, \tag{6}$$

where $d_{(n,k,i)}$ is $d_{(k,i)}$ decomposed at row n of **X**. All other unselected coefficients are then reset to zero. Figure 7 shows resulting **X** after selection of four coefficients in our example. Following the recipe in [21], the number of Haar wavelet coefficients used in our performance evaluation is heuristically set to $\min(L-1, \lfloor \log_2 L \times 4 \rfloor)$.

(iv) Reconstruct **X** with the above coefficients (c.f., Figure 8 for our example).

(v) Reduce the size of **X** by sampling down a group of repetitive and consecutive elements at each reconstructed row to one element. Figure 9 shows the reduced **X** in our example.

Note that the DTW distance between the reduced time-series matrices **X** of players i and j is assigned to the ijth element of the CMDS input matrix **D** discussed in Section 2.1.

4. EVALUATION

4.1. Settings

We obtained player log from the online game The ICE [23], under development at our laboratory. A screen shot of The ICE and the game map in use are shown in Figures 10 and 11, respectively. The main game objects were nonplayer characters (NPCs), statically positioned at different locations, with whom player characters (PCs) must interact—(chat, help, trade)—to receive and complete missions; the item-shop, from which PCs bought items and monsters, randomly positioned throughout the game world for PCs to attack with snowballs. Major missions in the game are as follows.

(i) Item delivery where the PC must deliver an item from the mission issuing NPC to a specified NPC.

(ii) Item trade where the PC must trade with NPCs to increase the amount of money initially provided by the mission issuing NPC.

(iii) Monster extermination where the PC must help the mission issuing NPC by exterminating monsters.

Actions available in The ICE are summarized in Tables 2 and 3. All NPCs are involved in missions, except NPCs 1, 13, 14, and 16. In the resulting KeyGraphs given in Section 4.2, the symbols for these four nonmission NPCs and monsters are preceded by "n" for those residing in Town 1, that is, nH, and "r" for those in Town 2 or the eastern border of the map, that is, rT, rU, rW, rA, and rD. This is done in order to utilize the visualization metrics in Section 2.3.

A group of 20 players, on average 20 years of age, participated in this evaluation. These players consisted of third-year and fourth-year computer science undergraduate students who were familiar with online-games but had no experience in playing The ICE. After a brief introduction

0.5	0.5	0	0	0	0	0.5	0.5
0.5	0.5	0	0	0	0	0.5	0.5
0	0	0.5	0.5	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0.5	0.5	0	0	0	0
0	0	0	0	0	1	0	0

Figure 8: Reconstructed **X**.

0.5	0	0	0	0.5
0.5	0	0	0	0.5
0	0.5	0	0	0
0	0	1	0	0
0	0.5	0	0	0
0	0	0	1	0

Figure 9: Reduced **X**.

Table 2: Action list of The ICE.

Action	Symbol
Attack with a snow ball	(c.f., top half part of Table 3)
Chat	c
Walk	w
Trade	t
Talk	(c.f., bottom half of Table 3)
Pick up potion	p
Use potion	u
Dead	d
Warp	x

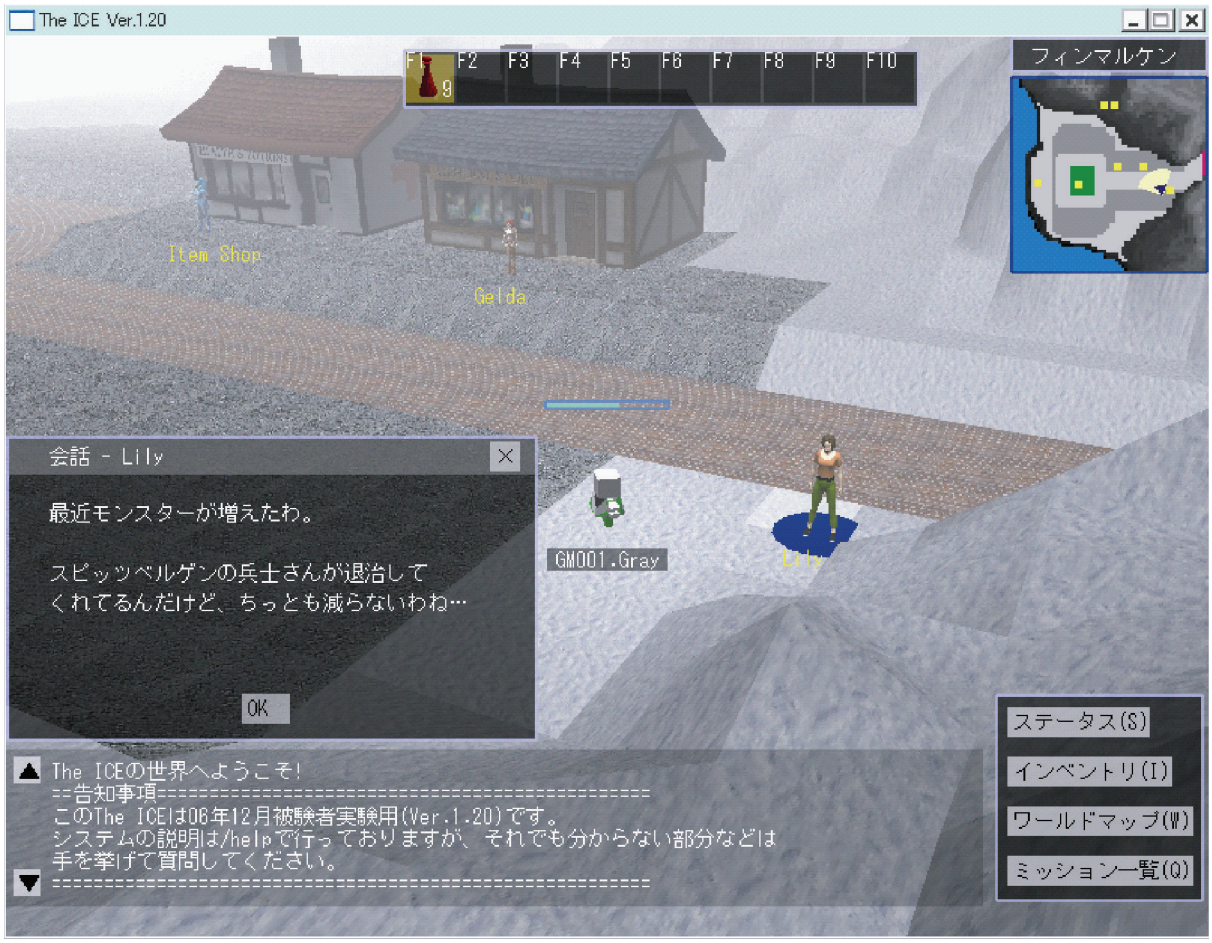


Figure 10: A screen shot of The ICE.

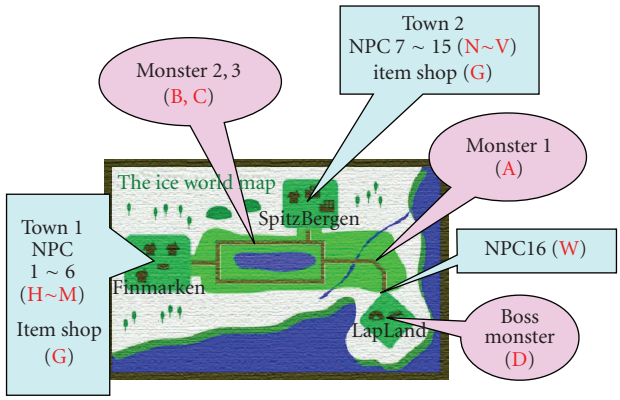


Figure 11: Map and the positions of NPCs and monsters.

Table 3: List of additional symbols related to actions Attack and Talk.

Symbol	Description
A	Attack to monster 1
B	Attack to monster 2
C	Attack to monster 3
D	Attack to boss monster
E	Attack to other game objects
G	Talk to the item shop
H–M	Talk to NPCs 1–6
N–V	Talk to NPCs 7–15
W	Talk to NPC 16

to the game, they were asked to arbitrarily play it, starting from Town 1. In addition to these 20 players, labeled p1–p20, three game masters, JOJO, Justice, KURO, also participated in the event. In the rest of our evaluation, the symbol w was removed from the log because it was frequently present in all players' action sequences and thus bared no information.

4.2. Results and discussions

Table 4 shows the mean and variance of time-series matrices of action sequences before and after the time-series reduction technique is applied.

Figure 12 plots all players on two-dimensional space obtained by CMDS. Most players form a cluster on the right half of the figure. The rests can be considered as outliers, that is, p1, p5, p8, p9, p17. To remove the effect of these outliers,

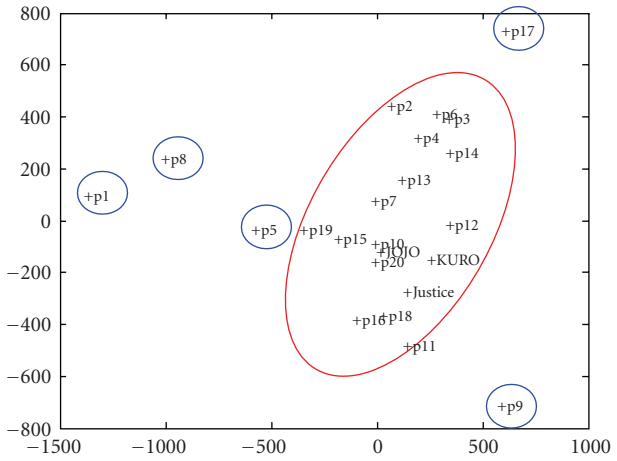


Figure 12: MDS result for all data.

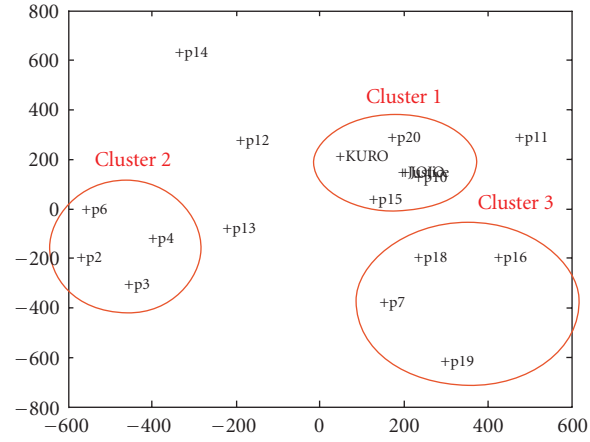


Figure 13: MDS result for data after exclusion of the outliers.

Table 4: Mean and variance of the data lengths before and after applying the time-series reduction technique.

	MEAV	VAR
Before	262.7	7688.6
After	25.5	13.1

we excluded them from the log and obtained a new result in Figure 13. From Figure 13, most players can be divided into three clusters: cluster 1 of Justice, JOJO, KURO, p10, p15, p20; cluster 2 of p2, p3, p4, p6; cluster 3 of p7, p16, p18, p19. Each cluster has different player behaviors as discussed below through interpretation of KeyGraph visualization results.

4.2.1. *Cluster 1: explorers*

Figure 14 shows the KeyGraph of cluster 1 from which salient features are summarized in the following.

(i) They moved away from town 1 and fought monsters 2 and 3.

(ii) They also went to the end of the map and fought the boss monster.

(iii) They were not active in pursuing missions.

The above summary is based on our interpretation of this KeyGraph as follows. First, it can be seen that the foundation of this KeyGraph is mainly composed of warp and attack (monsters 2 and 3) nodes. Next, the symbol rD is a keyword indicating that these players went far away to the end of the map and fought the boss monster there. In addition, because there is only one NPC symbol J in the keywords, these players were not active in receiving missions, from NPCs, and in pursuing them.

Consequently, it can be stated that the players in cluster 1 like to explore the world map and that these players have no interest in pursuing missions and only fight monsters when they find them. This type of players fits Bartle's explorer.

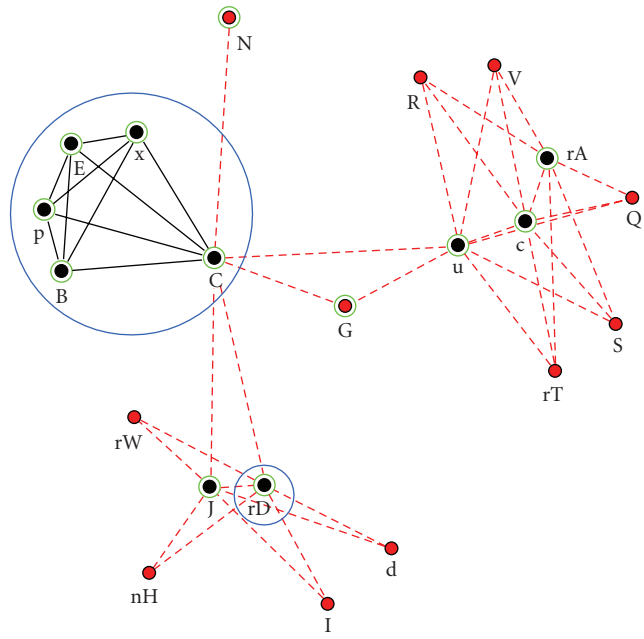


Figure 14: KeyGraph of cluster 1, where rD (attacks the boss monster residing in the most remote map from the initial map) is one of the keywords.

4.2.2. *Cluster 2: achievers*

Figure 15 shows the KeyGraph of cluster 2 from which salient features are summarized in the following.

(i) They mainly moved within town 1.

(ii) They also fought monsters 2 and 3.

(iii) They received a lot of missions.

The above summary is based on our interpretation of this KeyGraph as follows. First, it can be seen that besides nodes related to fighting (B, C, E, u, p), nodes of NPCs residing in town 1 (I, J, K) are included in the foundation of this KeyGraph. This indicates that these players were mainly in town 1. In addition, the keywords include symbols L and R which denote NPCs who are involved in several missions.

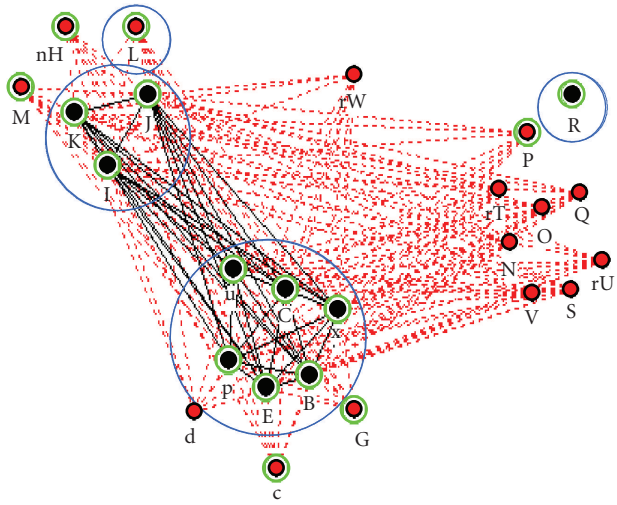


Figure 15: KeyGraph of cluster 2, where L and R (talk to mission NPCs) are among the keywords.

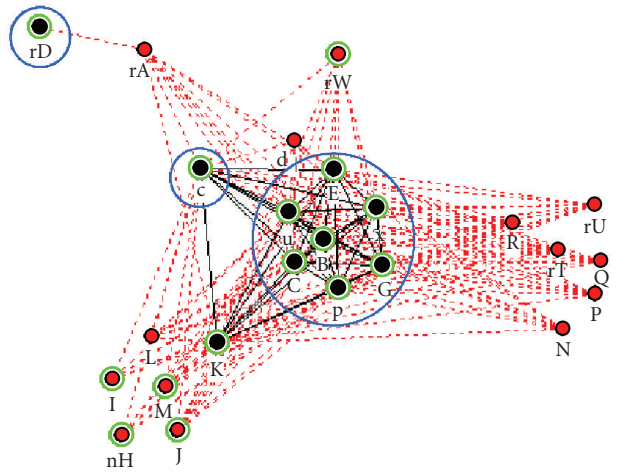


Figure 16: KeyGraph of cluster 3, where c (chat) is one of the keywords on the foundation.

As a result, it can be stated that the players in cluster 2 are aggressive in pursuing missions, especially those completable within or not far away from town 1. This type of players fits Bartle's achiever.

4.2.3. Cluster 3: socializers

Figure 16 shows the KeyGraph of cluster 3 from which salient features are summarized in the following.

(i) They chatted a lot.

(ii) They mainly fought monsters 2 and 3.

(iii) They also fought the boss monster.

The above summary is based on our interpretation of this KeyGraph as follows. First, the foundation of this KeyGraph includes the symbol c, not seen in the foundation or keywords of the previous two clusters. This indicates that these players chatted a lot among each other. Next, the symbol rD is a keyword showing that the players also fought the boss monster. We have confirmed through directly investigating the log that a group of three players (p5, p7, p8) and another group of four players (p16, p17, p18, p19) frequently chatted among their group members and that each group went together to the end of the map to fight the boss monster.

From the above interpretation, it can be stated that the players in this cluster like to communicate with others via chats. This type of players fits Bartle's socializers.

4.3. Computational complexity

We give here the computational complexity of the techniques used in our approach.

(i) *CMDS*: for m players, the time complexity of the original CMDS is $O(m^3)$. To cope with very large m, a recently proposed approximation [24] taking an $O(m)$ time can be used.

(ii) *DTW*: the time complexity of DTW for computing the distance between two time series of length l_x and l_y is $O(l_x l_y)$. We coped with this issue with the time-series reduction technique in Section 3. This technique can also be used together with an approximation technique in [25] that introduces lower bounding based on warping constraints.

(iii) *KeyGraph*: the time complexity of KeyGraph is $O(n \log n)$, where n is the number of action symbol types.

(iv) *Wavelet*: for a time-series of length l, the Haar wavelet transform has an $O(l)$ time.

5. CONCLUSIONS AND FUTURE WORK

Understanding the player behaviors is an important issue in improving the service quality of online games. We have proposed a visualization approach that first locates clusters of players who have similar action behaviors using CMDS and then interprets such behaviors of a cluster of interest using KeyGraph. To increase the efficiency in computation of the CMDS input, we have described the use of the time-series reduction technique proposed recently by us in [11]. Evaluation of the proposed approach has been done using log from The ICE, where three clusters have been found to fit three of the four Bartle's player types, that is, achievers, explorers, and socializers.

Our future work is to apply the proposed approach to log from commercial online games and to examine if Bartle's player types can be found. It might also be interesting to investigate log formats whose information can be used for automatically identifying other types of Nick Yee's play motivations.

ACKNOWLEDGMENTS

This work was supported in part by Grant-in-Aid for Scientific Research (C) no. 20500146 from the Japan Society for Promotion of Science. The authors would like to thank the anonymous reviewers for their invaluable comments.

REFERENCES

[1] P. Harding-Rolls, "Western World MMOG Market: 2006 review and forecasts to 2011," Screendigest Management Report, Screen Digest, London, UK, March 2007.

[2] K. B"orner and S. Penumarthy, "Social diffusion patterns in three-dimensional virtual worlds," *Information Visualization*, vol. 2, no. 3, pp. 182–198, 2003.

[3] N. Hoobler, G. Humphreys, and M. Agrawala, "Visualizing competitive behaviors in multi-user virtual environments," in *Proceedings of the 15th IEEE Visualization Conference (VIS '04)*, pp. 163–170, Austin, Tex, USA, October 2004.

[4] L. Chittaro, R. Ranon, and L. Leronutti, "VU-Flow: a visualization tool for analyzing navigation in virtual environments," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 6, pp. 1475–1485, 2006.

[5] R. Thawonmas, M. Hirano, and M. Kurashige, "Cellular automata and Hilditch thinning for extraction of user paths in online games," in *Proceedings of 5th ACM SIGCOMM Workshop on Network & System Support for Games (NetGames '06)*, Singapore, October 2006.

[6] R. Thawonmas, M. Kurashige, and K. T. Chen, "Detection of landmarks for clustering of online-game players," *The International Journal of Virtual Reality*, vol. 6, no. 3, pp. 11–16, 2007.

[7] R. Bartle, "Hearts, clubs, diamonds, spades: players who suit MUDs," *The Journal of Virtual Environments*, vol. 1, no. 1, 1996.

[8] M. Wattenberg and J. Kriss, "Designing for social data analysis," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 4, pp. 549–557, 2006.

[9] I. Borg and P. Groenen, *Modern Multidimensional Scaling: Theory and Applications*, Springer Series in Statistics, Springer, New York, NY, USA, 2nd edition, 2005.

[10] Y. Ohsawa, N. E. Benson, and M. Yachida, "KeyGraph: automatic indexing by co-occurrence graph based on building construction metaphor," in *Proceedings of the IEEE International Forum on Research and Technology Advances in Digital Libraries (ADL '98)*, pp. 12–18, Santa Barbara, Calif, USA, April 1998.

[11] R. Thawonmas and K. Iizuka, "Haar wavelets for online-game player classification with dynamic time warping," *Journal of Advanced Computational Intelligence and Intelligent Informatics*, vol. 12, no. 2, pp. 150–155, 2008.

[12] F. D. Fracchia, "Is visualization struggling under the myth of objectivity?" in *Proceedings of the 6th IEEE Visualization Conference (VIS '95)*, pp. 412–415, IEEE Computer Society Press, Atlanta, Ga, USA, October-November 1995.

[13] P. J. Somervuo, "Online algorithm for the self-organizing map of symbol strings," *Neural Networks*, vol. 17, no. 8-9, pp. 1231–1239, 2004.

[14] J. D. Cohen, "Highlights: language- and domain-independent automatic indexing terms for abstracting," *Journal of the American Society for Information Science*, vol. 46, no. 3, pp. 162–174, 1995.

[15] Y. Ohsawa and P. McBurney, Eds., *Chance Discovery—Foundation and Its Applications*, Springer, New York, NY, USA, 2003.

[16] R. Thawonmas and K. Hata, "Aggregation of action symbol sub-sequences for discovery of online-game player characteristics using KeyGraph," in *Proceedings of the 4th International Conference on Entertainment Computing (ICEC '05)*, F. Kishino, Y. Kitamura, H. Kato, and N. Nagata, Eds., vol. 3711 of *Lecture Notes in Computer Science*, pp. 126–135, Sanda, Japan, September 2005.

[17] N. Okazaki and Y. Ohsawa, "Polaris: an integrated data miner for chance discovery," in *Proceedings of the 3rd International Workshop of Chance Discovery and Its Management in conjunction with International Human Conputer Interaction Conference (HCI '03)*, pp. 27–30, Crete, Greece, June 2003.

[18] A. Tveit, Ø. Rein, J. V. Iversen, and M. Matskin, "Scalable agent-based simulation of players in massively multiplayer online games," in *Proceedings of the 8th Scandinavian Conference on Artificial Intelligence (SCAI '03)*, pp. 80–89, Bergen, Norway, November 2003.

[19] W. White, C. Koch, N. Gupta, J. Gehrke, and A. Demers, "Database research opportunities in computer games," *SIGMOD Record*, vol. 36, no. 3, pp. 7–13, 2007.

[20] N. Yee, "Motivations for play in online games," *Cyberpsychology & Behavior*, vol. 9, no. 6, pp. 772–775, 2006.

[21] F. K.-P. Chan, A. W.-C. Fu, and C. Yu, "Haar wavelets for efficient similarity search of time-series: with and without time warping," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 3, pp. 686–705, 2003.

[22] C. C. Aggarwal and P. S. Bradley, "On the use of wavelet decomposition for string classification," *Data Mining and Knowledge Discovery*, vol. 10, no. 2, pp. 117–139, 2005.

[23] The ICE Online Game, http://www.ice.ci.ritsumei.ac.jp/mmog.html.

[24] J. Tzeng, H. Lu, and W.-H. Li, "Multidimensional scaling for large genomic data sets," *BMC Bioinformatics*, vol. 9, article 179, 2008.

[25] C. A. Ratanamahatana and E. Keogh, "Three myths about dynamic time warping," in *Proceedings of SIAM International Conference on Data Mining (SDM '05)*, pp. 506–510, Newport Beach, Calif, USA, April 2005.