Evolutionary Optimization of Parameters for Distributed Virtual Environments

Simon E. Parkin, Peter Andras, and Graham Morgan

***Abstract*—In a distributed virtual environment interest management schemes attempt to ensure virtual world objects exchange messages only if they are interacting. This provides an opportunity to allow such systems to scale to support many hundreds, even thousands, of virtual world objects by reducing the need to send and process unnecessary messages. However, arriving at the optimum configuration for the interest management scheme is a challenging and complex problem. The customary approach is to configure parameters in an ad-hoc manner (supported by experience). However, such an approach is unlikely to yield optimum system performance. Here we propose a means of finding optimal parameter values by way of a simulation tool equipped with evolutionary optimization capabilities. We provide a series of experiments that indicate that our simulator can aid in configuring interest management schemes to gain better system performance.**

I. Introduction

DEVELOPMENT of scalable distributed virtual environments (DVEs) is a significant research challenge. Such environments must be engineered so as to be able to support many hundreds and thousands of users while satisfying requirements for timely performance (preserving responsiveness in virtual worlds) and consistency (ensuring participants view events in a mutually consistent fashion). With message exchange being the only way to propagate events that occur in the virtual world to a geographically dispersed network of users, care must be taken to achieve these requirements in a manner that avoids exhaustion of available networking and processing resources.

Interest management is an approach to building scalable DVEs (e.g. [1-4]) by making best use of available networking and processing resources via targeted message exchange (in contrast to a general broadcast mechanism). Messages are sent only to recipients that may be interested in them (e.g., messages are only sent between those objects that are interacting within the virtual world). The sending of messages is delegated across bounded areas within the virtual world (i.e., a virtual world object should be in the area to which a message is deemed applicable in order to receive that message). These areas are commonly termed *areas of influence*, and an object is expected to exert influence (i.e., send messages) to all other objects that exist in its area of influence.

Restricted message passing in a DVE based on areas of influence may lead to what are termed *missed interactions*: objects that should interact do not due to a lack of messages exchanged between them. Missed interactions occur because the degree on inconsistency in a DVE is sufficient to allow an object's traversal of an area of influence to go undetected, partially or fully, by an interest management scheme. Such inconsistency arises from the fact that delays in propagating events throughout a DVE are inherent due to networking and processing resource limitations (e.g., network latency). The missed interaction problem may be considered an interest management oriented view of the consistency-throughput trade-off described by Singhal and Zyda [5].

Assuming that networking and processing resources may not be easily altered by a developer of a DVE, there are three parameters within a DVE that may be manipulated to minimize the occurrence of missed interactions: (1) frequency of message exchanges; (2) object velocities; (3) sizes of areas of influence.

Earlier work [6] has shown that a tool capable of simulating a DVE allows a developer to experiment with a number of different parameter values to determine how best to minimize missed interactions while avoiding unnecessarily high message exchange frequencies (inhibiting scalability). However, such a tool is manually based and requires a developer to change parameters incrementally and run numerous individual experiments to gain an indication as to what parameter values may be appropriate. In addition, given that such a tool allows for the modeling of a wide variety of virtual world types (e.g., different styles of object movement, varying message exchange frequencies, and varying sizes of areas of influence) there are too many experimental combinations to run manually in a reasonable amount of time.

In this paper we describe how evolutionary optimization techniques have been employed to aid in deriving suitable parameters for governing a DVE. We take as the starting point the existing, manually based, DVE simulator tool [6]. We enrich the DVE simulator with an evolutionary optimization component that can determine optimized

Manuscript received March 13, 2007. This work was funded by UK EPSRC Grant GR/S63199 and UK EPSRC Grant GR/S04529/01.

Simon E. Parkin is with the School of Computing Science, Newcastle University, Newcastle upon Tyne, NE1 7RU, UK (e-mail: s.e.parkin@ncl.ac.uk).

Peter Andras, is with the School of Computing Science, Newcastle University, Newcastle upon Tyne, NE1 7RU, UK (e-mail: peter.andras@ncl.ac.uk).

Graham Morgan, is with the School of Computing Science, Newcastle University, Newcastle upon Tyne, NE1 7RU, UK (e-mail: graham.morgan@ncl.ac.uk).

settings for parameters that govern the functionality of a DVE. The results that we present confirm the applicability of our tool to the design of DVEs and shows that evolutionary optimization can be applied successfully to find optimum values for configuration parameters within a DVE.

The rest of the paper is structured as follows: In Section II we describe the role of interest management in designing scalable DVEs and the missed interaction problem associated with these techniques; Section III provides an overview of the DVE simulator; Section IV discusses the use of evolutionary optimization in the context of finding optimal parameters for DVEs. Section V describes a series of experiments that evaluate the evolutionary component of the simulator and in Section VI we draw the conclusions of our work.

II. Background and Related Work

In this section we describe the missed interactions that may be possible given different approaches to interest management. Such a discussion is presented in more detail in [6]. However, we present an abridged version to ensure the reader's understanding of the work presented here. After discussing missed interactions with respect to different approaches to interest management, a brief introduction to evolutionary optimization is presented. Although this field of research is well known to many and has been actively researched by many over a long period of time, it was thought beneficial to introduce the reader to this technique as some members of this paper's target audience (DVE researchers and developers) may not be familiar with evolutionary techniques. Finally, this section concludes with justification as to why an evolutionary approach to determining DVE parameters is desirable. This conclusion identifies the contribution of the paper.

A. Interactions and Interest Management

Interest management is the term used to describe how influence and interest exerted by objects in a virtual world translates to object interaction. Origins of interest management can be traced back to the early 1990's [1, 2], with a comprehensive introduction to the area provided by Singhal & Zyda [5]. There are many different ways of identifying areas of influence and interest in a virtual world for use in interest management (many examples given in [5]). Rather than provide a description of the many different approaches to interest management available, we shall concentrate on the two basic categories of interest management from which all others are derived. We deem this appropriate as a starting point for tackling the missed interaction problem in order to make our work sufficiently generic as to be applicable to past, as well as recent, approaches to interest management solutions (e.g., [3,4]).

All interest management schemes originate in *region based* or *aura based* approaches to defining areas of influence. In the region based approach (e.g., [2]) the virtual world is divided into regions of equal size and shape that remain static within the world. Messages are only exchanged between objects if they share the same, or neighboring, regions. In the aura based approach (e.g., [1]) each object has an aura associated with it (typically a sphere) that defines an area of the virtual world over which the object may exert influence. An object communicates its actions only to objects that fall within its aura or, possibly, with those objects that share aura overlap (e.g. [7]).

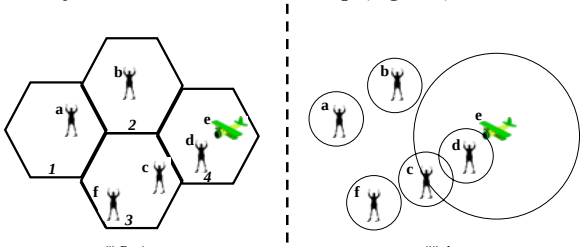

Fig.1. Areas of influence.

Identifying which objects are interacting at any one time in the region based approach can be achieved via a server that identifies which region participants are in and forwards messages appropriately. Alternatively, a communications subsystem capable of identifying multiple recipients of a message may be used. For example, each region may be assigned an IP-multicast address. If an object, say Obj_a, traverses a region boundary, say region 1 to region 2, Obj_a will subscribe as a sender/receiver for region 2's IP-multicast address and unsubscribe from region 1's IP-multicast address. Objects need only be aware of which region they are in to enable message passing to occur. There is no need for individual objects to contact each other directly to determine message recipients.

Due to the lack of static regions in the aura based approach, identifying message recipients is via the objects themselves (peer-to-peer) or via a server. In a peer-to-peer approach all objects must exchange messages periodically to realize when aura influence is exerted over objects. Such messages are commonly termed heartbeat messages and occur infrequently to avoid exhausting available bandwidth and message processing resources. Once aura influences have been determined, via a heartbeat message, then high frequency message exchange between objects may be enacted. In the peer-to-peer approach this is commonly achieved independently between two objects: if an object, say Obj_a, receives a heartbeat message from Obj_b, and determines that Obj_b is influencing Obj_a then Obj_a will make it known that it is interested in receiving Obj_b's high frequency messages. In the server approach to aura based interest management all objects send high frequency messages to a server, which assumes responsibility for determining interaction via aura influences. Once a server determines the appropriate influences then high frequency messages may be relayed between objects via the server.

Missed interactions may manifest themselves in a number of ways depending on the type of deployment scenario

(server or peer-to-peer) and the type of interest management scheme used (region or aura). We consider a period of unbroken interaction between objects to be a *session*. Missed interactions may be *complete* (throughout a session no messages were exchanged as expected) or *partial* (throughout a session a smaller number of messages were exchanged than expected). A server must inform objects of sessions taking place before missed interaction occur. As all messages are sent between objects and a server at a high frequency (irrelevant of region or aura based approaches), missed interactions relate to the speed (scalability) of the server in determining interaction.

In a peer-to-peer based solution missed interactions in the region based approach are related to the time taken for objects to realize within which area of influence they are in and the frequency within which they send messages. Avoiding missed interactions in a peer-to-peer deployment using aura based interest management is most challenging as the additional issue of regulating heartbeat message intervals needs to be considered: too infrequent and too many missed interactions will occur, too frequent and too many messages may exhaust available processing and networking resources.

B. Evolutionary Optimization

Evolutionary optimization [8,9] is a popular approach for solving complex problems for which it is difficult or impossible to find a reasonable solution through analysis.

Evolutionary optimization proceeds by creating successive generations of problem solutions, and improving the quality of solutions within consecutive generations through directed adaptation and selection of solution parameters. The quality of a given solution is determined by its *fitness*. This measure is determined by applying a *fitness function* to each candidate solution, resulting in a value that gives an indication as to how well the solution solves a given problem. With each problem that is to be assessed, the fitness function that is to be applied must be specifically tailored, so as to take into account all of the factors that contribute to the appropriateness of candidate solutions in providing a solution to a problem. The application of fitness functions to complex problem spaces then allows for the computational evaluation of potentially multi-parameter functions: functions wherein a number of contributing factors (or outputs) would need to be balanced when attempting to find an optimum solution, and for which an ideal solution is not intuitively apparent.

Candidate solutions within a problem space can be altered with successive generations. Progressive *adaptation* from one generation of candidate solutions to the next happens through evolutionary operators like *crossover* (the exchange of parts of the encoding of prospective solutions) or *mutation* (random alterations to the encoding of solutions). The *selection* operators simulate natural style evolution by identifying the fittest individuals and having them reproduce, passing components of their encoding (commonly termed *genes* in keeping with the evolutionary flavor of the context) to the next generation. As has already been stated, finding appropriate parameter values for a given DVE is a complex problem without a clear analytical solution, which makes it a good candidate for the application of evolutionary optimization techniques.

III. Simulator

This section describes our simulator. Only a brief description of the simulator is provided for completeness (described in detail in [6]). Detail has instead been afforded to the descriptions of the evolutionary component of the simulator with a detailed description of how the evolutionary technique is applied in section IV.

In developing our simulator we chose a peer-to-peer aura based interest management approach as this provides the most challenging environment when satisfying missed interaction and scalability requirements (see description in sub-section II.A).

A. Virtual World Model

We want to simulate an environment that is similar to a virtual world supported by a DVE application. This requires simulating objects that are capable of movement within a three dimensional space. We assume that there are distinct classes of objects residing within a virtual world, and that a particular object's movement is dependent on its object type with respect to these classes.

In terms of modes of movement, we define four types of object: **Direct** – objects move, without stopping, along a linear path at a fixed velocity; **Indirect** – objects move along a linear path at a fixed velocity, but may periodically deviate from their paths; **Stuttering** – objects move along a linear path at a fixed speed (when moving), but may pause for periods of time; **Static** – objects do not move.

In our simulations there are no obstacles, allowing objects to roam freely without hindrance. This aspect of our simulation ensures that measurements do not reflect any constraints that would otherwise have been imposed by the virtual environment itself (such as space limitations due to obstacles). The measurements gained from experiments using the simulator give a clear indication of the occurrence of missed interactions due to networking and processing constraints, neglecting any influence that virtual world obstacles would otherwise have had. Although including obstacles may give rise to interesting avenues for research in DVEs and have been explored in the real world (e.g., [10, 11]), such research is beyond the scope of this paper.

The simulator includes a virtual world parameter interface that allows the control of a subset of parameters on the basis of object class: lower and upper velocity bounds and the percentage of objects in the simulation that will be of a particular object type. Other parameters may also be set that relate to the virtual world as a whole: world size, number of objects within the world, the frequency of high frequency

and heartbeat message exchange, and values associated to networking resources (e.g., latency, jitter, bandwidth, processing availability).

We simulate meaningful (i.e., directed) movement of objects within the virtual world via the positioning of targets. Objects are assigned a target to move towards at random, and may change the target to which they are traveling towards in a similarly unpredictable manner. Given that the number of targets is controlled so as to be less than the size of the object population, with suitable constraints placed on the degree of randomness inherent in target reselection, objects will appear to cluster and disperse throughout the simulation.

The configuration information for a particular simulation can be stored to a data file which can be subsequently loaded back into the simulator for the purpose of reevaluating stored sets of parameters.

As we do not want resources of the machine actually running the simulator to influence our results we base time on the number of iterations progressed in a simulator. For example, if message latency is set to 3, then this indicates latency will last for 3 iterations of the simulation.

B. Evolutionary Component

The simulator supports the derivation of optimized simulation parameters by the use of evolutionary optimization techniques. Such techniques are applied to a fixed size population of simulated DVEs configured with a full range of static parameters (shared across the entire DVE population) and variable parameters (unique to each DVE in the population). As the evolutionary component of the simulator proceeds new generations of DVEs are created from the existing population using a range of genetic operators (crossover, mutation, and elitism). These operators are used to inform the process of selecting most promising candidate solutions. This optimization process stops when the optimum solution is discovered or after the generation of a user specified number of solution populations.

The evolutionary optimization component provides a visual representation of the evolutionary process in terms of charts displaying the performance measures and parameter values resulting from simulation runs. This data can aid the DVE developer in directing the search for optimized parameters in domains of the solution space where the desired level of performance is more likely to be achievable. After a specified number of solution generations have been created and evolved (or an optimum solution has been found), the simulator stops running simulations and compiles the results into a pair of linked charts.

In Figure 2 each dot represents a candidate solution (unique simulation). A point is positioned based on the heartbeat interval and aura size parameter values that were used during a simulation. The quality of the candidate solution is also illustrated: the darker the point, the better the associated parameters are at reducing the occurrence of missed interactions while keeping message exchange at a minimum (more accurately satisfying the fitness function). Due to printing limitations variable darkness is not clearly discernable in this paper. However, these graphs are presented to give the reader an indication of how the simulator works and what type of interfaces the simulator provides to users. The darkness (i.e., quality) of each point is determined with respect to all other simulations in the series, and may not necessarily be an 'ideal solution'. These charts are used to provide developers with a comparison for quick evaluation to aid in judging the success of an experiment.

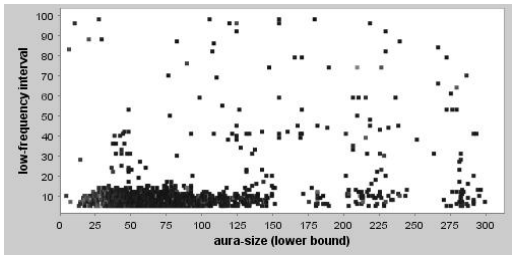

Fig.2. Aura size against heartbeat interval

In Figure 3 each dot represents the number of missed interactions that occurred during the simulation and the number of messages produced during the simulation.

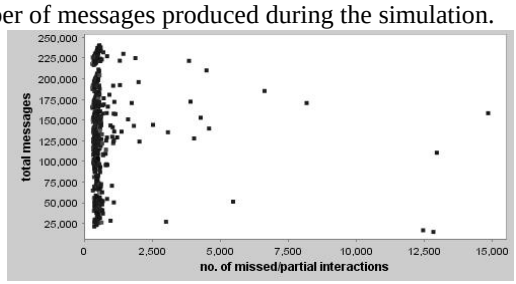

Fig.3. Missed interactions against messages

Figures 3 and 4 are linked, in that when a user highlights a specific point in either chart, the associated point in the adjoining chart is also highlighted. This allows a developer to identify which solution they want to use based simply on its quality (i.e., the darkness of the associated point), but also on its levels of message production or performance with respect to avoidance of missed interactions. This is especially applicable in situations where the demands of the developer vary. For example, a developer may be running a DVE application across a high performance network of nodes wherein the number of messages produced is less critical to system performance than the occurrence of missed interactions. This approach reduces the need for a developer to continually adjust the fitness function to suite their requirements (allowing quick and easy use of the tool for developers not familiar with evolutionary techniques).

IV. EVOLUTIONARY OPTIMISATION

In this section we describe how we have applied evolutionary optimization in our simulator. An overview is provided that describes a typical execution of the simulator

using the optimization component. This is followed by descriptions that highlight how functions and operators associated to the evolutionary component are managed in the simulator.

A. Overview

The simulator searches for optimum solutions by initializing a set of simulated DVEs with a complete set of configuration parameters. Heartbeat message interval and aura size (across all objects) for each DVE simulation are then modified in the search for optimum solutions. Each individual DVE simulation is run to completion, with performance observed with respect to missed interactions and the number of messages exchanged. The performance data from the simulation is evaluated by a fitness function to determine the effectiveness of the values used for aura size and heartbeat message interval. This concept of fitness also influences how subsequent generations of simulated DVEs are formed from the existing DVE set. We assume that the virtual world parameters relating to velocity, number of objects, size of virtual world and different types of objects remain the same throughout the series of experiments: given a particular virtual world we are attempting to determine the optimal settings for aura size and heartbeat message exchange.

Candidate solutions are encoded as individual *chromosomes* (common terminology for describing solutions in evolutionary optimization). These chromosomes include value encoded representations of the heartbeat message interval and aura size value pair to be trialed in each simulation. There is no need to encode any other values into the chromosomes as all other parameters are global in nature (i.e., shared by all simulations across all generations).

B. Fitness Function

As each complete set of simulation parameters is assessed, the effectiveness of specific parameter values at reducing the occurrence of missed interactions is evaluated using a specialized fitness function. This function is given as:

$$F = 1/3(1-C) + 1/3(1-P) + 1/3(A/E)$$

F is the overall fitness of the candidate solution; C is the percentage of missed interactions that occurred; P is the percentage of partially missed interactions that occurred; A is the number of messages sent during the simulation; E is the number of messages that it is estimated would have been sent during the simulation if no missed interactions had occurred.

The fitness function attempts to balance the number of missed interactions against the number of messages exchanged. If this were not the case and missed interactions were considered alone, it would be feasible to give preference to a DVE within which no missed interactions occurred. This may result in an unacceptably high number of heartbeat messages (hindering scalability). Using this function the closer F (the result of the fitness function) gets to a value of 1.00 the better the candidate solution is in striking the necessary balance between missed interactions and the number of messages exchanged. A simulation can only achieve a fitness of 1.00 if no missed interactions (complete or partial) occur, and the number of messages exchanged during the simulation is equal to the predicted number of messages that would have been exchanged by the system had no missed interactions occurred.

A developer may alter the fitness function. As already mentioned, the fitness function described here balances scalability (number of messages exchanged) against missed interactions (partial and complete). However, a small change to the fitness function may be made by a developer to ensure complete missed interactions are avoided at the expense of partial missed interactions. Alternatively, different parameters may be incorporated into the fitness function to determine how many different types of a particular object may influence missed interactions and/or scalability. Tailoring of the fitness function in this manner is not without complications (e.g., may gain misleading results) and requires a developer who understand fully how evolutionary techniques may be applied in the context of DVEs.

C. Crossover, Mutation, Elitism

With the fitness of each candidate solution derived, the *mating potential* of each chromosome can be determined with relation to all other candidates in a generation. This is achieved, in our approach, by comparing fitness values across the population in conjunction with the standard deviation of all the fitness values. The most promising chromosomes are more likely to mate (i.e., blend their characteristics into a hybrid offspring) or alternatively live on into the next generation.

Selected pairs of candidate solutions are chosen to mate as each new generation in the optimization process develops, thereby blending their characteristics into an offspring chromosome. This is achieved primarily by use of the *crossover* technique. Using this technique two solutions are chosen based on their mating potential. Heartbeat message interval and aura size parameters are randomly selected from one or either of the two parent solutions, forming a single composite offspring (with the hope that mating two good chromosomes will produce a better one). Furthermore, in order to maintain a certain level of variety in the overall set, there is a chance that random, small mutations (of varying but bounded magnitude) are created in the offspring chromosomes. This may or may not contribute to better candidate solutions in the following generation(s).

In order to preserve candidate solutions which show the most promise (in as far as them being prospective parents in subsequent generations), all chromosomes with a fitness value above the average within the existing population are allowed to ‘live on’ into the next generation. This is one form of *elitism*, wherein the best chromosomes are chosen to

outlive the rest of the generation. Elitism prevents the loss of good solutions once they are found and helps the speed-up of the evolutionary optimization process [12].

A small subset of the new population is generated from heavily mutated offspring spawned by chromosomes from the previous generation that were deemed below average in quality. This is an additional step taken to ensure that the solution space does not go stale, by essentially giving a second chance to those chromosomes that wouldn't have lived on. This step also affords the capacity to quickly search out candidate solutions in other parts of the solution space as a whole (by mutating offspring chromosomes away from any refined solution spaces currently under investigation). If this heavy mutation was excluded from our simulation it may be quite conceivable that an ideal solution may go undetected as the solution space narrows through subsequent generations.

V. Performance

In this section we describe a series of experiments that were carried out to evaluate the evolutionary component of the simulator. The main focus of the experiments was to determine how variations in the evolutionary techniques themselves would contribute to determining optimum parameters. To this end, in the majority of our experiments virtual world parameters were left unchanged. This allowed comparisons to be made when altering evolutionary related settings (e.g., how soon is an optimum value reached).

A. Simulator Settings

We assessed the usefulness of our simulator using a base set of global configuration parameters, detailed as follows: World size – 5 000; No. of iterations – 500; No. of objects – 50; No. of targets – 2; Generation limit – 100; Networking latency – 2; Processing latency – 1. We also maintained a base set of parameter values across all object classes for all simulations: Object class quotient – 25; Lower velocity bound – 10; Upper velocity bound – 20; High frequency interval – 5. We carried out a series of experiments to evaluate various aspects of the simulator (see sub-sections below). To calculate the average values we run each experiment 50 times with random initialization for each setting of the investigated parameter.

To ascertain whether the simulator was effective a set of measurements from the experiments was recorded: average fitness and variance of all the chromosomes in each generation. These measurements show how the solution set changes over time, arriving at an optimum set of parameter values. In essence, we are concerned with not only the suitability of the optimized solution, but how long it took to derive and the variability present in the candidate solutions.

B. Population Size

The simulator was tested with a small set of different population sizes, specifically 10, 30, 50, 70 and 90 chromosomes. The results for these tests are shown in Figures 5 and 6.

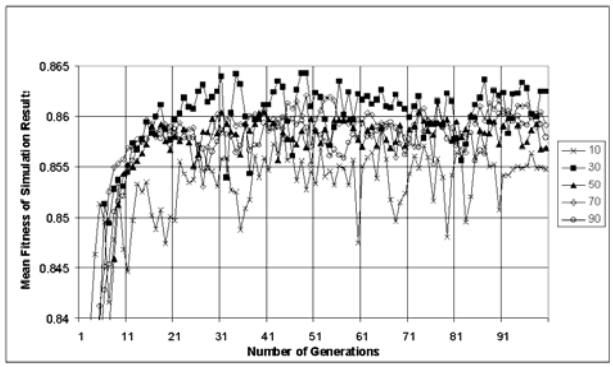

Fig.4. Generations against average fitness of chromosomes for different population sizes

Figure 4 shows that for all population sizes the simulator quickly starts to converge on higher quality solutions, and for the most part is able to pursue these promising solutions. At least with respect to our own DVE environment, increasing the population size above 30 chromosomes does not necessarily affect the capacity to retain viable solutions, as the average fitness remains relatively uniform in such instances. However, a population size as low as 10 chromosomes shows frequent and sizeable fluctuations in the average fitness of the population, suggesting that a population this small would be unsuitable for retaining promising candidate solutions.

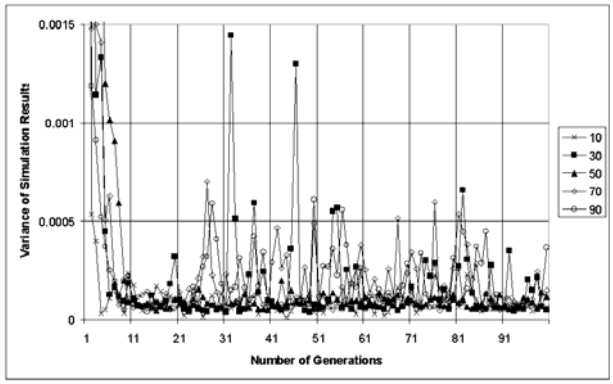

Fig.5. Generations against variance across chromosomes for different population sizes

Figure 5 shows how the variance in the solution set for all instances is quickly reduced as each instance progresses. This illustrates that the simulator is able to focus on promising solutions to increase the quality of the population in general and is able to maintain this level of quality throughout each run. The results for all population sizes show infrequent spikes in variance, attributed to the simulator searching out new solution spaces using heavily mutated offspring; it is noted, however, that despite these spikes the general level of variance is kept constant, which again proves that the simulator is able to refine the solution space and maintain improvements made to the solution set as a whole.

C. Mutation

Mutation allows the examination of specific solution spaces by essentially fine tuning the solution set with minor mutations (in the case of offspring created from elite chromosomes), or alternatively quickly search out other promising solutions through the heavy mutation of offspring generated from non-elite chromosomes. Different levels of mutation in the elite offspring were examined, to determine how such alterations affect the ability to find and improve upon promising solutions. The level of mutation is determined by both the probability of mutation occurring and the maximum achievable extent of any mutations that occur (represented as a quotient of the overall parameter range). As such, we examined a small set of mutation levels: probability of 5% with a bound of 1%; probability of 15% with a bound of 1/75th; probability of 25% with a bound of 2%; probability of 40% with a bound of 5%; and finally a probability of 50% with a bound of 10%. The results of these tests are shown in Figures 7 and 8.

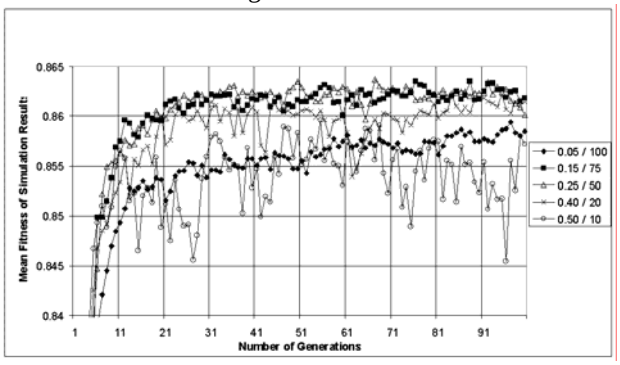

Fig.6. Generations against average fitness of chromosomes for different levels of mutation

Figure 6 shows that for a mutation level of 5% with a 1% bound the average fitness quickly converges on a set of promising solutions. However these values result in the simulator been slow to fine tune the solution set, steadily increasing the average fitness albeit with no dips in overall quality. With the two most extreme levels of mutation it is evident that the solution set in general suffers lasting (if not necessarily severe) dips in fitness, which may affect an ability to retain and fine tune promising solutions. Therefore, at least for the DVE problem we are solving, it appears that a chance of mutation of around 15-25%, with a bound of $1/75^{th}$-$1/50^{th}$ of the parameter range is the most suitable choice.

Figure 7 shows that for the lesser three mutation levels variance is kept relatively low (once the values have quickly converged upon a minimum), but that for the two most severe levels of mutation the variance gradually becomes pronounced in its fluctuation. This suggests that the ability of the simulator to retain and fine tune its solution set is hampered by the chosen levels of mutation in these cases. The mutation levels otherwise help the simulator to augment the candidate solutions in other cases. Considering this outcome, it is again recommended that mutation levels are kept relatively low.

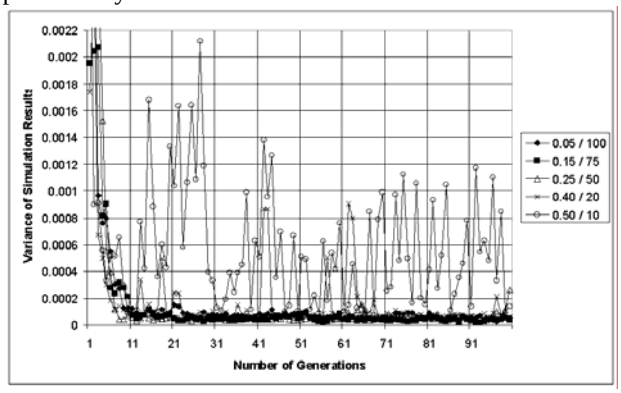

Fig.7. Generations against variance across chromosomes for different levels of mutation

D. High Frequency Message Interval

In order to test the simulator's capacity to find optimum solutions given variability in one of the fitness functions parameters, we simulated DVEs with the same base world parameters but with varying high frequency messaging intervals. The specific test set ranged across high frequency messaging intervals of: 1 iteration; 2 iterations; 5 iterations; and 8 iterations.

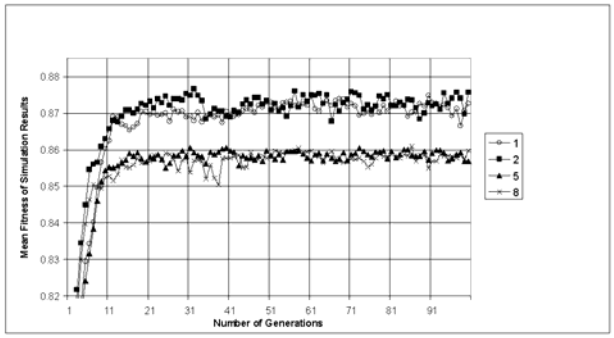

Fig.8. Generations against average fitness of chromosomes for different high-frequency messaging intervals

From the average fitness chart for high-frequency messaging intervals (Figure 8), it is evident that the simulator is capable of quickly converging on promising solutions across the different scenarios. In addition, the graph indicates that the simulator is able to find more promising solutions as the high frequency interval is decreased. For intervals of 1 and 2 iterations the average fitness is relatively equal, with the same being said for the pairing of 5 and 8 iteration intervals. For the smaller pair of values in this test set, these results suggest that the simulator is capable of balancing message production with the occurrence of missed interactions as the capacity for lower message frequencies did not necessarily yield better results with respect to the tests for 1 and 2 iteration high frequency messaging intervals.

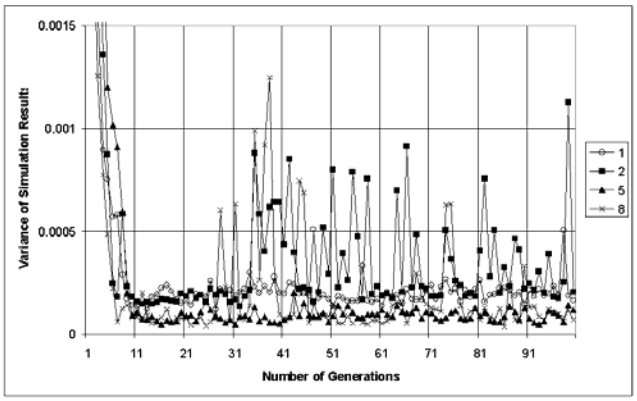

Fig.9. Generations against variance across chromosomes for different high-frequency messaging intervals

Figure 9 shows how the simulator quickly fine tunes the solution set for all instances that are investigated and maintains a relatively consistent level of variance throughout each run. There is minimum difference between the various intervals measured with respect to variance. This indicates that high frequency message exchange probably plays a lesser role in overall messages exchanged compared to heartbeat messages. This was also clear in our previous experiments (described in [6]).

VI. CONCLUSIONS

Distributed virtual environments play an increasingly important role in networked computer games, persistent virtual worlds, and advanced scenario training (such as military simulations). Handling the consistency throughput tradeoff is critical in such environments, and interest management is a key approach to delivering effective and efficient system.

We have presented (as an extension to our existing suite of DVE applications) a novel optimization tool, in the form of a DVE simulator, that allows the experimental evaluation of candidate configuration parameters for DVEs. The simulator builds a simulation of the proposed DVE and uses evolutionary optimization to find optimal values for the parameters that mostly influence a DVEs consistency (minimizing missed interactions) and scalability (minimizing message exchange). The experiments described in this paper indicate that evolutionary optimization techniques do provide appropriate solutions for DVEs in a manner that would be either extremely time consuming or impossible to achieve manually.

We expect that our work will facilitate the systematic addressing of problems arising in the context of DVE development. The simulator allows for the analysis of the impact upon DVE performance of particular sets of configuration parameters. Incorporating evolutionary optimization makes it entirely possible to deal with the highly nonlinear and complex problem of determining optimal values for DVE parameters (not simply consistency and scalability as described here). We believe that the combination of simulation and evolutionary optimization to be the way forward for the objective development and assessment of DVE designs.

Our next step is to incorporate our middleware monitoring software [13] into the simulation tool presented here. In addition we aim to provide appropriate changes to our DVE simulator parameters to accommodate changes in networking and processing resources and virtual world properties during runtime.

The complete suite of DVE simulation tools may be found at: homepages.cs.ncl.ac.uk/graham.morgan. The middleware monitoring software is also accessible through this website.

REFERENCES

[1] Greenhalgh, C., Benford, S., “MASSIVE: a distributed virtual reality system incorporating spatial trading”, Proceedings IEEE 15th International Conference on distributed computing systems (DCS 95), Vancouver, Canader, June 1995

[2] Zyda, M. J., Pratt, D. R., “NPSNET: A 3D visual simulator for virtual world exploration and experience”, in Tomorrow’s Realities Gallery, Visual Proceedings of SIGGRAPH 91, (1991) pp. 30

[3] Knutsson, B. , Lu, H., Xu, W., Hopkins, B., “Peer-to-Peer Support for Massively Multiplayer Games”, The 23rd Conference of the IEEE Communications Society, 2004

[4] Bharambe, A., Pang, J., Seshan, S., “A Distributed Architecture for Interactive Multiplayer Games”, In CMU CS Technical Report Number CMU-CS-05-112, 2005

[5] Singhal, S., Zydra, M., “Networked Virtual Environments, Design and Implementation”, Addison Wesley, 1999

[6] Parkin, S. E., Andras, P. , Morgan, G., “Managing Missed Interactions in Distributed Virtual Environments”, in the Proceedings of the 12th Eurographics Symposium on Virtual Environments, Lisbon, Portugal, 8th - 10th, ACM pp. 27 - 34, May 2006

[7] Morgan, G., Lu, F., “Predictive Interest Management: An Approach to Managing Message Dissemination for Distributed Virtual Environments”, In Proceedings of the First International Workshop on Interactive Rich Media Content Production: Architectures, Technologies, Applications, Tools (Richmedia2003) 2003

[8] Fogel, D. B.: An Introduction to Simulated Evolutionary Optimization. IEEE Trans. on Neural Networks: Special Issue on Evolutionary Computation, Vol. 5, No. 1, pp. 3-14, 19

[9] Goldberg, D. E., “Genetic Algorithms in Search, Optimization and Machine Learning”, Addison-Wesley Longman Publishing Co, 1989

[10] Brocklehurst, D., Bouchlaghem, D., Pitfield, D., Palmer, G. and Still, K., “Crowd circulation and stadium design: low flow rate systems”, Structures & Buildings, Volume 158, Issue: 5, October 2005

[11] Helbing, D., Buzna, L., Johansson, A., and Werner, T., “Self-Organized Pedestrian Crowd Dynamics: Experiments, Simulations, and Design Solutions”, Transportation Science, Vol. 39, No. 1, February 2005, pp. 1-24

[12] Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T., “A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II”, IEEE Transactions on Evolutionary Computation, Vol.6., No.2, pp.182-197.

[13] Morgan, G., Parkin, S. E., Molina-Jimenez, C., Skene, J., “Monitoring Middleware for Service Level Agreements in Heterogeneous Environments”, In the Proceedings of the fifth IFIP conference on e-Commerce, e-Business, and e-Government (I3E 2005), IFIP Volume 189 pp. 79-93, 2005