INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

VFC-RTS: Vector-Field Consistency para Real-Time-Strategy Multiplayer Games

Manuel Eduardo Costa Cajada

Dissertation for the Degree of Master of
Information Systems and Computer Engineering

Jury

President:	Prof. Mário Rui Fonseca dos Santos Gomes,
Supervisor:	Prof. Luís Manuel Antunes Veiga,
Co-Supervisor:	Prof. Paulo Jorge Pires Ferreira,
Member:	Prof. Daniel Jorge Viegas Gonçalves

May 2012

Acknowledgements

For starters, I would like to thank Prof. Luís Veiga, for all his support and guidance through the development of this project, pushing me to continue when I got stuck and encouraging me to take risks. Additionally, I would like to thank him for correcting my english during the writing of this document.

I would also like to thank all my friends and apologize for my absence during the numerous invitations that I couldn't attend. A special thanks to Sta, who was always prepared to take some time to help me whenever I needed. Thank you.

I could not fail to mention Lurdes Oliveira, the person who most put up with me and my lousy temper during this hole process, always by my side, motivating me to carry on, not letting me give up. Thank you for everything.

Finally, I want to thank my sisters and parents for always believing in me. I doubt I would be writing this thesis if it were not for your dedication and support. Thank you for making me the man that I am today.

Lisboa, July 11, 2012
Manuel Cajada

to José & Maria

Resumo

Apesar dos jogos online no ambiente multi-jogador em massa terem vindo a ganhar muita popularidade ao longo dos anos, os jogos de estratégia em tempo real (RTS) não foram considerados fortes candidatos para a utilização deste modelo devido ao número limitado de jogadores suportado, tendo em conta as topologias de rede normalmente utilizadas, ao grande número de entidades em jogo e aos fortes requisitos de consistência.

Para lidar com esta situação, conceitos como *continuous consistency* e *location-awareness* provaram ser extremamente úteis para restringir áreas com requisitos de consistência. O primeiro conceito procura estabelecer um equilíbrio entre consistência e disponibilidade ao definir zonas de consistência com limites de divergência. O segundo toma em consideração a localização do jogador durante o processo de decisão, sustentando que os eventos ocorrendo mais perto da sua localização apresentam um maior interesse para decisões futuras. A conciliação dos resultados destes dois conceitos resulta numa técnica poderosa em que a localização do jogador e limites de divergência estão directamente ligados, fornecendo ao jogador informações precisas sobre os objectos dentro da sua área de interesse. As técnicas de gestão de interesse existentes apresentam uma abordagem de tudo ou nada relativamente à consistência, o que resulta em perdas de visibilidade e presença intermitente de objectos nas fronteiras da área de interesse do jogador, proporcionando a este uma experiência de jogo confusa e irritante.

Uma abordagem diferente para este problema é apresentada no modelo Vector Field-Consistency (VFC). O modelo VFC alcança um melhor equilíbrio entre as noções de *continuous consistency* e *location-awareness* através da utilização de várias zonas de consistência com diferentes limites divergência em torno da localização do jogador (*pivot*). Neste trabalho apresentamos o VFC-RTS, uma adaptação do modelo original VFC, caracterizado pelo estabelecimento de graus de consistência, para o cenário dos jogos RTS. Isto torna este trabalho especialmente relevante uma vez que, nos jogos RTS, os mapas de jogo (por exemplo, campos de batalha) tendem a ser muito grandes, populados por grandes números de entidades (soldados, tanques, edifícios, etc) e os jogadores estão envolvidos em várias localizações (como exemplo, frentes de batalha).

Começamos este trabalho por fazer uma revisão geral sobre o estado da arte nas áreas de manutenção da consistência, gestão de interesse e jogos de estratégia em tempo-real. Seguidamente, apresentamos a nossa solução para o sistema VFC-RTS, descrevendo como os conceitos do modelo VFC original foram adaptados ao paradigma RTS, propondo ainda uma arquitectura para um *middleware* genérico. Posteriormente, o modelo VFC-RTS é aplicado a um jogo RTS *open-source* e multi plataforma com requisitos de consistência total. Por fim, foram analisados os resultados da aplicação da *framework*, avaliando o sucesso do modelo, expondo seus pontos fortes e possíveis melhorias.

Abstract

Although massive multiplayer online games have been gaining most popularity over the years, real-time strategy (RTS) has not been considerate a strong candidate for using this model because of the limited number of players supported for the usually employed network topologies, large number of game entities and strong consistency requirements.

To deal with this situation, concepts such as continuous consistency and location-awareness have proven to be extremely useful in order to confine areas with consistency requirements. The first concept seeks a balance between consistency and availability by defining consistency zones with divergence boundaries. The second takes into consideration the player's location during the decision process, defending that events occurring closer to a player's location present bigger interest for future decisions. The combination between these two concepts results on a powerful technique in which the player's location and divergence boundaries are directly linked, providing the player the most accurate information about objects inside his area-of-interest. Existing interest managing techniques apply an all-or-nothing approach regarding consistency, presenting visibility loses and intermittent object presence near the player's area-of-interest boundaries, creating a confusing and annoying experience for the user.

A different approach to this problem is presented in the Vector-Field Consistency (VFC) model. The VFC model achieves a better balance between the notions of continuous consistency and location-awareness by defining multiple zones of consistency around the player's location (pivot) with different divergence boundaries. In this work we propose VFC-RTS, an adaptation of the VFC model, characterized for establishing consistency degrees, to the RTS scenario. This is especially relevant since in RTS games, game scenarios (e.g., battlefields) tend to be very large, populated with large number of game entities (soldiers, tanks, buildings, etc.) and players are engaged in multiple locations (e.g., battlefronts).

In this work, we start by making an overview about the state-of-the-art in the fields of consistency maintenance, interest management and RTS games. Then we present our solution for the VFC-RTS system, describing how the concepts of the original VFC model were adapt to the RTS paradigm, and propose an architecture for a generic middleware. Later, The VFC-RTS model was applied to an open source, multi-platform RTS game with full consistency requirements. Finally, we retrieve the results of the application of the framework and evaluate the success of the model, exposing its strong points and possible improvements.

Palavras Chave
Keywords

Palavras Chave

Vector-Field Consistency

Jogos de Estratégia em Tempo Real

Modelo de Consistência Contínua

Consciência da Localização

Jogos Multi-Jogador em Massa

Gesto de Interesse

Keywords

Vector-Field Consistency

Real-Time Strategy Games

Continuous Consistency Model

Locality-Awareness

Massive Multiplayer Online Games

Interest Management

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the increase of the internet services capabilities over the years (i.e. larger bandwidth and packet throughput) there has also been a growth in the number and variety of applications. Among these new types of real-time applications, one that has been gaining most popularity are massive multiplayer online games (MMOG). Games of this class distinguish themselves from other kinds of online games by supporting a great number of players sharing a virtual world, possibly very large, while interacting with each other and with the surrounding environment.

In the world of the MMOGs, the category most played and with the greatest popularity are the role-playing games (MMORPG). Typical examples are titles such as *EverQuest*[1] and *World Of Warcraft*[2]. In these games, the participants play the role of a character in a virtual world, experiencing it through the eyes of an avatar. There are other real-time network game categories which have been less discussed for using the MMOGs model, among them the real-time strategy games (RTS)[21]. In this class of games, the player manages a set of entities, usually varying between different kinds of structures and pawns, with the objective of commanding the troops throughout the map until achieving total control over it.

Multiplayer solutions for RTS games typically allow a limited number of participants to be playing simultaneously (in the order of dozens)[7] due to the lack of scalability of the traditionally used network architecture, the Client-Server (CS) model. This architecture is characterized by the use of a centralized server that is responsible for maintaining the game state by collecting updates from all clients, resolving inconsistencies and propagating the most current game state. Other network models have already been considered to deal with the scalability issue, among them the Peer-to-Peer (P2P) model[12, 13]. In a P2P architecture, each peer is both a client, responsible for executing its own game simulation[27], and a server for other peer clients. Although this results in decentralized resource consumption and an increase of game resources every time a node joins the system, it is most costly in bandwidth consumption because it takes more messages to synchronize game state between all participants, resolve inconsistencies and coordinate overall game progress. Hybrid solutions have also been presented that balance the trade-off between performance and communication costs, like Peer-to-Peer with Central Arbiter (PP-CA)[27] and Proxy-Server[21].

When playing over the internet, users are subjected to internet service conditions that may not always satisfy the game's QoS (Quality of Service) specification. Latency is often the greatest network related

[1] http://everquest.station.sony.com
[2] http://www.worldofwarcraft.com

issue because it affects the usability of the games, compromising users QoE (Quality of Experience). On a FPS (First-person shooter) the latency value generally tolerated has to be less than 50 milliseconds[6]. However, for a RTS, the latency can achieve higher values (up to 200 milliseconds) without affecting the game.

"Latencies up to several seconds have little effect on the final outcomes of building, exploration, and most combat. Overall, strategy plays a much larger role in determining the outcome of the game than does latency" [7]

Maintaining game state consistency on a RTS game is a complex task. Not only does a player have to be aware of the progress of his troops throughout the map (considering troops in the order of hundreds or even thousands), but also maps on a MMOG tend to have very large dimensions. In addition, the system has to be prepared to deal with changes in network quality without losing performance. To reduce the amount of data on a state update process and still provide the user with the relevant state information, MMOGs employ *interest management* (IM) techniques.[5, 40]. This concept, commonly used on other MMOG classes of games like RPG[2, 5] and FPS[19], restricts the amount of data messages during the update process by limiting the area of interest (AoI) of a player. Thus, the player only receives the game state data necessary to notice changes on the scenery up to a certain range, mostly corresponding to the area where avatars and events relevant to him are located and take place.

In spite of reducing the bandwidth consumption, existing IM techniques present an all-or-nothing approach, providing the user with every update about objects inside his AoI and non about objects outside. This situation can result on visibility losses, intermittent presence and movement of objects near the AoI boundaries, and decrease on playability.

There are techniques to deal with changes in network conditions without compromising the consistency. *Dead reckoning* for instance, reduces the bandwidth usage by sending update packets less frequently. By using a prediction algorithm the system can compensate information gaps, avoiding an inconsistent state. Other techniques like packet compression or aggregation are used to minimize network traffic and bandwidth requirements respectively[32].

1.1 Contributions

This work proposes the adaptation of a continuous consistency model to the needs of the RTS games. To achieve it, we start by identifying the consistency requirements of real-time strategy and how the continuous consistency model can meet them without compromising the dynamic of the game. Later, we analyze how the concept of area-of-interest can be combined with the continuous consistency model so that, with a partial view of the game state, the player is, at all times, provided with the information necessary to make immediate decisions.

In Figure 1.1, we illustrate examples of total and partial game state update, respectively. On total game state update, each player receives updates regarding all entities present on the game, not taking into consideration their position on the map neither the update's relevance to a player's current strategy, while, on partial updating, each player only receives updates concerning entities inside his area-of-interest.

Although the first solution prevents divergences between players by propagating the full game state, this operation is very expensive to the network since it requires the broadcast of all game updates, and inefficient since a player may receive updates about entities that, located outside his area-of-interest, will

Figure 1.1: Total vs. Partial game state update

not influence the decision making process. The second solution bounds the update propagating to the entities within a player's area-of-interest, thus greatly reducing the number of updates exchanged. By combining the notions of continuos consistency and location-awareness we expect to notice an increase of game scalability which can be translated in the increase of the maximum number of player on a single session, game entities controlled by the player or even of the maximum number of units on map, without compromising game performance or the player's QoE.

The model implemented is the *Vector-field Consistency* (VFC) model[29, 38]. Designed specifically for the gaming environment, VFC presents its own IM technique, characterized by the use of *consistency zones*, presenting increasing consistency requirements as the distance from the player's area-of-interest decreases.

1.2 Results

The main result derived from this work is the adaptation and optimization of the Vector-Field Consistency model to the real-time strategy environment: Modifying the premises of the VFC model in order to adapt it to the reality of the RTS games, improving the system scalability without compromising availability, performance or playability. Also resultant from this work we produced:

- A study, analysis and evaluation of the state-of-the-art, researching the topics of consistency models, IM techniques and MMOGs, also including network architectures.
- The concretization of the solution as a *middleware* and application of the result to *Warzone2100*[3], an open source, futuristic RTS game.
- An evaluation of the success of the solution, determining if it improves the scalability of the game through an analytical study, using the metrics presented on the *GSM: Game Scalability Model*[20, 21]. To complement the evaluation, a qualitative study is also performed to evaluate the impact of the solution on the player's perspective.

[3]http://wz2100.net

1.3 Document Roadmap

Next, in the Chapter 2 we explore the main topics around consistency maintenance, reviewing existing solutions and how they apply to the gaming scenario, followed by an overview on the major interest management techniques and other techniques to reduce bandwidth requirements. Finally, we study network topology as well as the different design choices when developing a real-time strategy game.

Then, in Chapter 3, we present in detail the architecture employed on the development of our solution, followed, in Chapter 4, by some of the most relevant aspects that emerged during its implementation.

In Chapter 5 we describe the evaluation environment and present the results obtained, analyzing them at the same time. The text finishes with Chapter 6, where we summarize the work performed, introducing our ideas for future work.

Chapter 2

Related Work

The following section addresses a study over three major topics to determine the state-of-the-art of Consistency in Distributed Systems focusing the RTS games environment. First, an overview on the general consistency scenario, exploring the different design choices and kinds of systems (section 2.1). Second, we present how general consistency concepts apply to the game environment (section 2.2). Finally, in section 2.3, we present the different architectural solutions for MMOGs, focusing on RTS games.

2.1 Consistency in Replicated Systems

In the field of Distributed Systems, high availability and performance are two major requirements assured by the use of data replication. By maintaining multiple replicas on different computers, a system can assure that the failure of one or more replicas will not cause the data to became unavailable. Furthermore, by allowing each user to connect to the nearest replica to access data, data replication also improves load balancing, thus improving performance. The issue of maintaining consistency in distributed systems can be described as the "ability to keep replicas sufficiently similar to one another despite operations being submitted independently at different sites"[28]. However, as there are multiple replicas there is also the possibility of information disparity between them, leading to inconsistency. In the next sections we will present two distinct classes of replication techniques: *pessimistic replication* (section 2.1.1) and *optimistic replication* (section 2.1.2), followed by a review of some well stablished systems that explore the consistency spectrum (section 2.1.3), and finally, in section 2.1.4, we presented an overview about the discussed topics, making an comparison between the presented solutions.

2.1.1 Pessimistic Replication

Pessimistic replication techniques aim at maintaining single-copy consistency, usually by usage of locking mechanisms[22]. Although users can read data from any of the existing replicas, as long as it is up to date, only one user at a time is allowed to perform write operations to a replica, blocking the remaining users during the process. These provide strong consistency by preventing conflicts between updates since data changes are isolated and propagated to all the other replicas, becoming available for all the following read operations.

Despite being widely used in commercial systems[14] due to performing well in local-area networks, this solution is not suited for the Internet, as intermittent connectivity could result on blockage of the system or even data corruption. In addition, pessimistic replication algorithms scale poorly because of the frequent update propagation, compromising performance and availability.

2.1.2 Optimistic Replication

Contrary to the pessimistic approach, optimistic replication algorithms[28] are based on the assumption that conflicts between updates occur only rarely and can be easily fixed when they happen. As a result, optimistic algorithms allow replica data accesses without previous synchronization between all replicas. Hence, less frequent communication is required, leading to better performance and availability. Additionally, optimistic replication increases flexibility, scalability and enables asynchronous collaboration, making it the ideal solution for wide-area or mobile environments[28].

Optimistic replication introduces the concept of *eventual consistency* which states that even though the state of replicas may diverge, the continuous propagation of updates in background will eventually lead to the full consistency of the system. While in pessimistic consistency the sequence of operations to an object in a site must be preserved when applied on the remaining sites, optimistic replication can support partially inconsistent data. Nevertheless, this property makes this model unsuitable for systems with strict consistency requirements that cannot tolerate occasional conflicts.

The design of an optimistic consistency system demands a number of choices regarding the properties expected from the application. The following portion of the paper presents an overview of these aspects and how they influence the outcoming system.

Number of writers

An optimistic system can be classified according to the number of writers, as **single-master**, where one replica is assigned the master and all updates are submitted by it, being then propagated to the remaining replicas, or **multi-master**, letting updates be submitted by any replica independently. Even though multi-master systems provide greater availability, they are much more complex as they have to be able to handle conflicts and scheduling, in opposition to single-master systems.

Definition of operations

This property takes into consideration how updates are disseminated from the replica where they originated to the other replicas. This process can be done by **state-transfer**, substituting an entire object every time a byte is modified, or **operation-transfer**, transferring the operations corresponding to the user changes, recreating the changes to the objects on the other sites.

State-transfer[29] is a simpler procedure, compared to operation-transfer, and can result in easier convergence of replicas in cases where the application of every missing update proves to be more costly than replacing the entire object. This proves suitable for situations when sites stay off-line for long periods of time. Although operation-transfer[30] is a more complex approach, since it requires maintaining the history of operations, it proves to be more efficient when objects are large and operations high-level, and more flexible for conflict resolution, reducing bandwidth requirements.

Scheduling

In order to achieve consistency, every replica has to produce the same schedule of event, resulting in a consensual final state. There are two different scheduling policies: **syntactic scheduling** which orders operations based on absolute information such as submission timing, location and issuer, and **semantic scheduling** which makes an effort to organize operations in an "optimal" way, exploiting semantic properties like commutativity with the goal of reducing conflict occurrence and rollbacks. Syntactic scheduling proves to be a simpler solution since it requires no decision making, disregarding relations between operations. However, this may cause the detection of unnecessary (or even false) conflicts, avoidable by such things as changing the order of execution of two concurrent operations, only possible with semantic scheduling.

Conflict Handling

One of the defining characteristics of a distributed system is how it handles the occurrence of conflicts between updates. Conflicts are detected when operations do not satisfy their prerequisites. There are several way to manage conflicts and they all are divided in two phases: **detection** and **resolution**. Pessimistic algorithms and single-master systems handle conflicts by preventing them from happening, restricting the number of users capable of submitting updates, one at a time in the former, and assigning one replica as the one writer of the system in the latter. This solution, however, affects negatively the availability of the system. Other systems handle conflicts by ignoring them, leading to an increase of lost updates and, consequentially, of difficulties in data management.

Conflict detection policies, like scheduling policies, are classified as **syntactic** or **semantic**. Syntactic conflict policies do not present explicit prerequisite specification, making use of happen-before relationship [16] to intuitively capture the relations between operations. Semantic conflict policies take advantage of semantic information about the operations to designate if a conflict has happened. Although syntactic conflict policies are more efficient than semantic policies they lack flexibility and incur in a higher number of false conflicts, due to the assumption of the happen-before relationship that any two concurrent operations are conflicting.

Regarding conflict resolution, there are two practices: **manual**, presenting to the user the two conflicting versions and letting him decide which will be the final one, or **automatic**, used in such systems as Bayou[36], which requires the definition of preconditions and merge procedures to each operation. Although this approach is most appealing and some times easily implemented (using time references or priority systems), studies prove that writing merge procedures can also be a very complex task.

Propagation Strategies and Topologies

Update propagation is a key element of an optimistic system, defining how and when updates are distributed among replicas. Propagation strategies can be categorized in terms of **communication topology** and **degree of synchrony**. Topologies with a fixed structure can be highly efficient but lack the dynamism needed for failure-prone and unstructured networks. These kinds of networks rely on epidemic propagation algorithms, allowing operation propagation even in networks with dynamic structure.

The degree of synchrony is a unit of measurement for the frequency and speed of inter-site communication and update exchange. The range of the degree of synchrony goes from pull-based systems, where a replica has to request updates from the remaining replicas, to push-based systems, where a replica with new updates proactively informs other replicas. To obtain the lowest degree of replica inconsistency and

rate of conflict, the general approach is to apply quick update propagation, compromising the complexity of the system.

Consistency guarantees

As the state of replicas may diverge, there are consistency guarantees that stipulate how much the state between replicas may differ. In the spectrum of allowed divergence there are two extremes: **single-copy consistency**, ensuring that the result of multiple writes from different sites is the same as a serialized execution of operations in a single site, and **eventual consistency** that guarantees only that the state of replicas will eventually converge. This concept offers weak assurance about when consistency among replicas will be reached, however, it is the common choice for optimistic-replication systems, designed to accomplish availability. Hybrid consistency models have been proposed which seek solutions between single-copy consistency and eventual consistency, often achieved by blocking access when consistency conditions are not met (divergence bounding)[41, 29] (section 2.1.3).

2.1.3 Case-study Systems

In this section we present a review of four case-studies, representing the evolution of the optimistic replication scenario. First, a mature technique by the name of *Operational Transformation* that applies the semantic scheduling schema, followed by three divergence bounding systems: *TACT*, *VFC*, and finally, *Data-aware Connectivity*.

Operational Transformation

Designed for real-time cooperative editing systems, operation transformation (OT) is a consistency maintenance technique that makes use of semantic scheduling.

The first system to implement the concept was the GROVE[10, 33] system, which used a replicated architecture, where each participant would have a local replica of a shared document and when a participant wanted to submit an update he would execute it immediately on the local copy and then propagate the update to remotes sites for execution. Before executing the update, the remote replicas apply a transformation to the operation, discarding the need to reorder previous operations. The GROVE system identified two major inconsistency problems regarding the execution of remote operations: *divergence* and *causality violation*. The problem of divergence emerges from the fact that, if a set of non commutative operations is not executed in the same order in every site, the execution will result in different final states. The causality violation problem can be easily detected in a situation where replicas execute received updates immediately after arrival, without respecting natural causal order.

With the previously announced problems as its basis, GROVE defines two properties as consistency correction criteria: **convergence property**, stating all generated operations have to be executed at all sites, and **precedence property**, stating if one operation a causally precedes another operation b, then at each site the execution of a happens before the execution of b. The solution presented by GROVE consists of two components: a *state-vector timestamping*[26] scheme to implement the precedence property and to ensure the convergence property the *distributed OPerational Transformation* (dOPT) algorithm. The dOPT algorithm performs a transformation to every operation that complies to the precedence condition against independent operations. The outcome of the transformation should be a set of operations that will produce the same final state when executed in any site.

More than a decade later, REDUCE[35] explored a third inconsistency problem designated *intention-violation problem*, related to the execution of a sequence of operations not producing the results expected by the user even though replicas are consistent. To deal with this problem, the authors of REDUCE defined the *CCI Consistency model*, stating that consistence is only achieved when the previous presented GROVE properties are contemplated plus the *intention-preservation* property: for any operation O, the effects of executing O at all sites are the same as the intention of O, while not changing the effect of independent operations. To ensure these properties, REDUCE applies a distinct approach to achieve convergence: an undo/do/redo schema and implements the *Generic Operational Transformation* (GOT) algorithm[34] to enforce intention-preservation, making use of pre and post conditions to capture the user intention when transforming operations.

GROVE was a pioneer in the field of operational transformation, opening doors for new systems such as Jupiter[24], which ported the operational transformation paradigm to the scenario of shared persistent virtual worlds.

TACT

In the spectrum of consistency, designers of replicated systems are normally forced to choose between one of the two extremes, pessimistic replication, associated with performance overheads and limited availability, and optimistic replication, characterized by eventual consistency. The TACT (Tunable Availability and Consistency Tradeoffs) framework[41] is a continuous consistency model that enables the designer to bound the consistency requirements of the application. Thus, it allows the designer to define the maximum semantic distance between replicas and even different consistency boundaries per-replica. When the operation meets the pre-established requirements it proceeds locally, otherwise it blocks the operation until synchronizing with other replicas as specified by the system's consistency requirements.

On TACT, an object consistency semantics are defined by *conits*, using three metrics: (1) *Numerical error*, which bounds the consistency distance between the local value of the conit and the value of the "final image", i.e an image of the system where all updates have been applied, (2) *Order error*, used to quantify the difference between updates order on the local replica and the ordering on the "final image" and (3) *Staleness* is the maximum delay allow between the current time and the acceptance time of the oldest write on a conit not seen locally. The implementation of the first metric is easily accomplished for applications with numerically identified operations, needing to specify the weight for each write operation otherwise. The numerical error is then the weight of all writes applied to a conit at all replicas unseen for the local replica. This value can be hard to obtain as it relies on the cooperation between all replicas, implying the use of a consensus algorithm to update it. Order error bounds the maximum number of writes that can be rolled back or reordered, making use of a write commitment algorithm and compulsory write pull to implement the order error bound. Staleness is maintained using a real-time vector that holds a entry for each replica stating the time passed since the last update.

TACT contributed to the consistency scenario by presenting a solution where different consistency levels can be assigned to different replicas. A replica allocated on a mobile device, for instance, which is subjected to poor network conditions and processing power constrains, may relax its consistency to reduce the incoming communication amount (updates generated on other sites), without compromising the consistency of the remaining replicas (outgoing updates remain unchanged). In the same way, a replica with great communication conditions and processing capability may impose strong consistency at little cost. The TACT framework take advantage of this property to increase the performance of the system by routing clients to replicas that meet their consistency requirements.

VFC

The Vector-Field Consistency model[29, 38] is, as the TACT framework, a continuous consistency solution that allows a per-replica consistency level specification. However, contrary to the TACT solution, it also allows to change consistency requirements dynamically based on the application's run-time state. Moreover, it applies the concept of *locality-awareness* to increase availability, user experience and reduce resource consumption. To implement *locality-awareness* it is necessary to establish one or more points (sets of coordinates specified by the developer, or dynamically by the application) from which is calculated the distance to the surrounding objects. This points are called the *pivot points*. A pivot point represents the position around which there are strong consistency requirements, weakening as the distance from the pivot increases. In order to define the consistency bounds, delimitating the authorized divergence degree, with the pivot as the center, there are drawn ring-shaped, concentric areas presenting increasing radius and divergence degree, called *consistency zones*. This design concept is further explored in section 2.2.2.

To formalize the consistency degree of a zone, VFC employes a 3-dimensional consistency vector: (1) *time*, the maximum time a replica can be without the last update values (2) *sequence*, the maximum number of lost updates supported, and (3) *value*, the maximum divergence between replicas or against a constant. The values chosen for the vector should produce a consistency guarantee so that it does not compromise the user experience by presenting bad information.

VFC proves to be an very flexible solution as it allows explicit and dynamic definition of consistency degrees, suitable for a wide variety of systems. Furthermore, it is developer-friendly thanks to its simple and intuitive settings parameterization schema, and player-friendly since it guarantees the player is provided with all the information needed to make sensible decisions. Also, VFC accomplishes an efficient management of the system resources by, intelligently, classifying updates with different priorities, sending critical updates immediately and postponing less-critical ones. Designed originally for the distributed games environment, the VFC model can easily be ported to other environments by adapting the concepts of pivot, *consistency zones* and distance to the target systems. One area where it has presented successful results is on the field of cooperative work tools[8].

Data-aware Connectivity

Intended for the mobile environment, the goal of the Data-aware Connectivity[3] (DaC) system rests on the principle of ensuring acceptable quality of a replicated object at minimal connectivity resources cost. To reach this goal, the system implements a continuous consistency model that enforces data consistency bounds of local replicas, along with a *connectivity monitor* that, using knowledge about the existing communication supports, calculates the less expensive connection to obtain the desired quality levels. When the system cannot exploit any connection successfully it refuses accesses to the replica. This solution proves to be greatly useful for mobile systems, where, nowadays, a larger number of network technologies are available, presenting different price costs, ranges and resources consumption levels, by intelligently choosing the best suited connection for the given task. Additionally, by leaving decisions regarding object refreshing and connection choosing to the system, not only it becomes more transparent to the user but also reduces error-prone user decisions.

The Data-aware Connectivity is composed of two modules: a *quality monitor* and a *connectivity regulator*. The quality monitor is responsible for continuously estimating the divergence degree of each replicated object. To calculate when to refresh an object, the quality monitor considers, among other criteria: (1) time since the last synchronization from each replica; (2) number of local tentative updates; (3) commit weight of currently missing from accessible replicas; (4) number of known concurrent updates;

	Systems			
	OT	TACT	VFC	DaC
Number of Writers	multi-master	multi-master	multi-master	multi-master
Dissemination	operation	operation	operation	state/operation
Topology	any	any	centralized	any
Degree of Synchrony	push	push/pull	push/pull	push/pull
Server Update Sending Order	when ready	when ready	priority	priority
Update Execution Order	reception	reception	reception	reception
Consistency Guarantees	eventual	bounded	eventual/bounded	bounded
CPU Load	high	high	high	medium
BW Load	low	low	low	very low
Mobile Support	no	no	yes	yes
Divergence Bounding	no	yes	yes	yes
Locality Awareness	no	no	yes	no

Table 2.1: Optimistic replication system's design choices

and (5) recent update activity by other replicas to the object. By contemplating these criteria, the quality monitor ensures the properties of freshness (1), consistency (2) and possibility of quick commitment (3-5).

Existing continuous consistency systems, such as TACT or VFC, ensure replica quality by forbidding access to replicas once they exceed the allowed divergence degree and until they are obeyed again. However, in this system when replica quality is below the bounds defined, it probes available connectivity in order to reestablish consistency levels, forbidding replica access only if no connection is possible.

2.1.4 Conclusions

From the design standards previously presented some conclusions can be taken regarding the kind of optimistic system intended. Single-master systems are well suited for applications with multiple readers or a single writer and with little conflict tolerance. Multi-master systems prove to be a better option for applications with multiple writers but rely on eventual consistency. State-transfer systems are simple to manage and simple to implement, presenting low memory and constant bandwidth requirements, what makes them adequate for most optimistic systems. Nevertheless, the fact that these systems cannot resolve conflicts between operations, as do operation-transfer systems, makes them unsuitable for applications with an elevated number of concurrent events, and for applications that may produce small modifications to files of big dimensions, compromising the network's bandwidth.

Table 2.1 summarizes the section by presenting the relevant optimistic replication design choices taken in the case-study solutions. By analyzing the table it is possible to point out the main differences between the systems:

(1) While OT and TACT are suited to any network topology, VFC, designed originally for small wireless network games, implies the use of a centralized architecture. More recent work has successfully applied the VFC model to a distributed architecture[38, 23].

(2) Contrary to the OT and TACT ordering solutions, where updates are sent as they are generated, VFC uses a priority system, sending critical operations first than less critical ones. Although this system implies the classification of each operation regarding its context and content importance, increasing the computational load, it is a small compromise compared to the reduction of bandwidth consumption.

(3) Regarding consistency guarantees, each system takes a different position: Although OT offers weak consistency guarantees and TACT establishes consistency boundaries by specifying the divergence degree allowed by the application, VFC presents a hybrid solution in which the area closer to a pivot point presents stronger consistency and as the distance from the pivot increases the consistency requirements go weaker, leading to, ultimately if desired, just eventual consistency.

2.2 Consistency in Games

One of the aims of MMOGs is to provide an environment where realistic interaction among players can occur. For that, every participant should be unaware of any data inconsistency regarding entities of the game. In this section, we present how the replicated distribution paradigm previously analyzed is applied to MMOGs reality (section 2.2.1), techniques to, in real-time, identify the AoI of the user (section 2.2.2), and techniques to minimize the network communication load, such as *Dead Reckoning* (section 2.2.3), packet compression and aggregation (section 2.2.4).

2.2.1 Adaptation and Usage of Classic Consistency Approaches

In this section of the paper we present how the different approaches for consistency management previously presented are used in the distributed multiplayer games domain.

Pessimistic Approaches

In order to maintain game state consistency in each participant, pessimistic algorithms make use of lock or flow control methods[22]. Lock based approaches are similar to *token ring* type network which allows data transmission for a node that has the token in the network. A user is allowed to manipulate a shared object exclusively when he is in possession of the token, sending the token to the next element of the ring afterwards, possibly with the updates piggybacked in the token message. Even though these approaches provide strong consistency and are simple to implement, transferring the ownership of the token among the nodes can be delayed due to network latency[25], compromising the user experience.

Optimistic Approaches

Optimistic consistency algorithms intend to maximize real-time interactivity among participants by removing the necessity of waiting for the ownership of shared objects. To accomplish this goal, the local game state is immediately updated and then transmitted to other participants by state or operation transfer. Although these approaches imply immediate responsiveness even in poor network conditions, when inconsistencies arise, state repair procedures are in order, which may increase the computational load and reduce playability. *Frequent State Regeneration*, *Bucket Synchronization* and *Time Warp* belong to the optimistic consistency maintenance approaches.

Frequent State Regeneration[31] is a state-transfer algorithm that propagates updates by regularly sending packets containing complete data regarding current frames in game sessions, disregarding the occurrence of state changes. Thus, the receiver can maintain consistency by identifying and repairing the divergence between the local state and the remote state. Games employing this algorithm include titles

like *Doom*[1] and *Diablo*[2]. This algorithm is simple to implement as it does not require central servers or lock management. Nevertheless, it scales poorly, as the number of objects in a session increase, and presents elevated bandwidth requirements.

The Bucket Synchronization mechanism, employed in the game *MiMaze*[17, 32], divides the game time in blocks of the same length and each block presents a "bucket", a set of updates. Remote updates, as well as local updates, are assigned to the appropriate bucket according to the prefixed playout delay. This delay determines how much time must be passed to display the results of user actions. In this way, a message issued at absolute time is delayed according the playout delay so that all participants can evaluate it at the same time. The data stored in the buckets are used to render game frames in sequential order. In case a message is missed or arrives too late, the algorithm uses data from the preceding bucket to extrapolate the information, using *Dead Reckoning* (section 2.2.3).

The Time Warp[39] algorithm adopts the virtual time paradigm, characterized by the use of a 1-dimension temporary coordination system for distributed processing management, exploiting the logical clock conditions proposed by Lamport[16]. On a normal execution, the algorithm processes any update (local or remote) that presents a timestamp (virtual time) larger than the last processed event, leading to a global ordering of events. However, if an arrived update presents a smaller timestamp than the latest one, a rollback procedure is triggered, canceling all events referring to updates not correctly ordered and processing the reordered queue of events. To cancel incorrectly ordered events, Time Warp sends an anti-message for each message previously flagged, which will cause the rollback of other processes. This procedure may cause degradation of the system's performance as well as flooding of the network with anti-messages.

Other consistency models, such as the continuous consistency model proposed in [18], approach specific areas of state consistency, in this case, object trajectory. This particular model proposes a *relaxed consistency* approach that relaxes the strict time-dependent consistency control on individual states of a replicated object, allowing the object to deviate its trajectory by an acceptable amount.

2.2.2 Interest Management

In order to provide a shared sense of space among players, each player must maintain a copy of the **relevant** game state on his computer. The simplest approach requires every player broadcasting updates to all other players so that each player can maintain a full copy of the game state. However, this solution is proved to scale poorly. IM techniques[2, 5] reduce the number of updates to be transferred to other players by utilizing filtering mechanisms such as application specific and/or network attribute specific. In this section we focus on application specific IM politics and classify a number of IM algorithms according to them.

Interest Management schemes are usually classified, according to the range of action, as based in: *aura-nimbus*, *range* or *visibility*.

Aura-Nimbus

The aura is the area that bounds the presence of an object in space while the nimbus corresponds to the area-of-interest of an object, the area in which an object presence can be perceived by the others[5].

[1] http://www.idsoftware.com/games/doom
[2] http://eu.blizzard.com

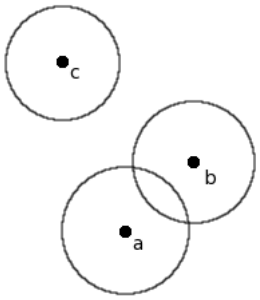


Figure 2.1: *Aura-nimbus* IM

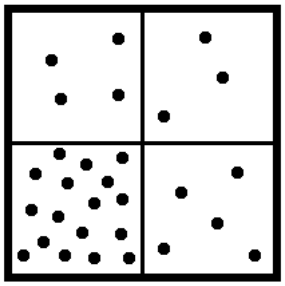


Figure 2.2: Region-based IM

In its simplest model both the aura and nimbus can be represented by fixed-size circles around the object (Fig. 2.1). Considering an observer object a and observable object b, when the aura and nimbus are coincided with each other, a can receive the information in regard to b. This model proves to be advantageous by allowing a fine-grained interest management, limiting the number of sent updates to only the relevant ones. However, it scales poorly due to the cost of computation associated with the identification of intersections between auras and nimbus.

Range

Used in such systems as RING[11] to mitigate the limitations of the aura-nimbus model, range-based interest management partitions the world space into static regions, assigning each object on a region to the AoI of every other object it the same region (Fig. 2.2). This model is often cheaper to compute than a pure aura-nimbus model. Nevertheless, the quality of the model is dependent of the shape and size of the regions along with the distribution of object among the regions. In a situation when several objects are concentrated in one region (illustrated in Fig. 2.2), load balancing mechanisms are often required. Although regular square tilings are the most popular choice, hexagons make a better approximation to the aura-nimbus model.

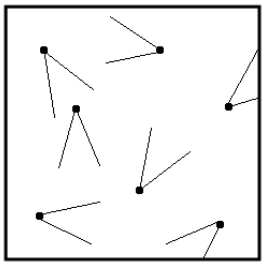


Figure 2.3: Visibility-based IM

Visibility

Contrary to the other IM approaches presented, visibility-based interest management considers the vision of the object/player instead of a fixed radius, taking in great consideration the geography of the game environment. The RING system, referred in the last model for applying a region-based approach, also makes use of the visibility model for precomputing visibility between regions[11], limiting the updates received by an object not only to its region but also to the range of its vision. The advantage of this technique is that the visibility of the objects is determined precisely and without additional cost since object information is already computed for rendering. However, this model assumes that a client has already received the information regarding all objects that may be visible in the world, or, in systems

like RING that also apply region-based IM, visible in its region.

To obtain the finest-grained update filtering, many systems opt to apply more than one IM model. One example of this is the already referred **RING** system. Designed to support a large number of players in a shared environment, the RING system applies both a range-based model, limiting the AoI of every object to the region they are positioned, and a visibility-range model, further restricting the interest zone to the area the player's visibility can reach.

A3[4], an algorithm intended for the MMOGs environment, combines the aura-nimbus and the visibility-based models in order to limit the AoI of the player to an area of 180 degrees in front of the *avatar*, ignoring objects localized in the back. This property may affect the user experience in situations when the *avatar* makes sudden rotation movements, causing a delay on the objects to become visible. To handle this situation in a less brusque manner, the algorithm also considers a second aura of smaller dimensions, retrieving information about only the closest objects behind the avatar. Moreover, A3 also includes a notion of *consistency degree*, requesting more often updates to objects closer to the *avatar* and decreasing the frequency of updates as the distance from the *avatar* increases.

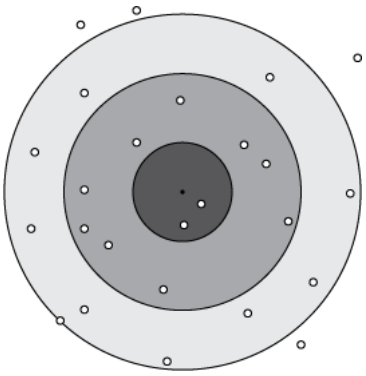


Figure 2.4: 2-dimensional VFC

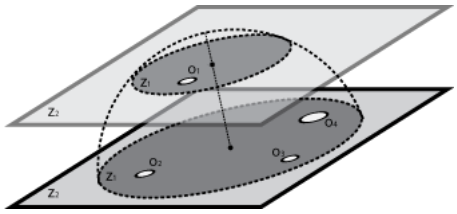


Figure 2.5: 3-dimensional VFC

VFC[29, 38], presented in section 2.1.3, applies multiple concentric aura each with different radius. This design complements the VFC consistency model, limiting *consistency zones* as rings around a *pivot* point (Fig. 2.4). This way objects situated in rings closer to the *pivot* point present greater consistency requirements which decline as the rings go wider. Moreover, the VFC interest management technique is also supported in a 3-dimensional environment (Fig. 2.5). This can be accomplished by porting the shape of the AoI to the 3D spectrum (concentric spheres). In the solution, objects situated within *consistency zones* on top of the *pivot* point present the same consistency requirements any other object in the same zone.

2.2.3 Dead Reckoning

In opposition to the application specific solutions previously discussed to reduce the bandwidth use, network specific approaches, such as Dead Reckoning[32], seek this goal by sending update packets less frequently. Especially effective when handling position information, Dead Reckoning makes use of approximation techniques to extrapolate the information of missing updates based on updates previously received (therefore leveraging semantic knowledge of the game and to some extent, location-awareness). Additionally, Dead Reckoning applies convergence algorithms to avoid abrupt movements when new location information is applied.

To achieve the most accurate approximation results, prediction techniques may employ such trajectory parameters as velocity and acceleration. The values of these variables can be obtained in two different ways: approximated from the history of last received updates or attached in each object's position update. Although the second way is more accurate, the communication overhead associated to the transmission of additional information leads to an increase of bandwidth consumption. Predicting the velocity and acceleration of an object proves to be less costly bandwidth wise, however, if new location information is not received through large periods of time, small inaccuracies in the algorithm may accumulate in significant errors.

When a node receives an update message, the object's predicted position is likely to differ from the position based on the latest information. To deal with this situation, Dead Reckoning makes use of convergence algorithms that vary in complexity. The simpler solutions, such as moving the object directly to the position on the latest update, have no additional computation requirements but produce jerky movements. More complex algorithms pick additional points between the two positions to produce a smoother movement but have greater computation requirements.

2.2.4 Packet Compression and Aggregation

Packet compression techniques can reduce average packet size by encoding packet header and/or body to be transferred to other nodes[32, 9]. In general, compression methods can be divided into two groups, ***lossless*** and ***lossy*** compressions. On the one hand, lossless compresions methods can restore original data from encoded one without any loss, usually capable of shrinking the size of data to approximately down to half. On the other hand, lossy compression algorithms are not required to recover the original state of compressed data. These algorithms are broadly utilized to compress multimedia data significantly compared to the lossless compression by exploiting the imperfection of human perceptibility such as visibility and audibility.

Packet compression methods can be categorized into **internal** and **external** types. Internal compression algorithms encode packets without referring previously transferred packets. This types is advantageous for the easiness of implementation and no requirement of reliable protocols, however, it presents a low compression rate. External compression methods can achieve high compression rate by utilizing packets transmitted beforehand but require storing of packet history. Furthermore, reliable transmission protocols are demanded to uphold the dependency of packets.

The bandwidth requirement can be alleviated by utilizing packet aggregation techniques. Aggregation techniques reduce the overhead of packet headers by combining multiple packets into one packet, consequently alleviating bandwidth requirements. The packet aggregation methods can be categorized into **timeout** and **quorum-based** types[32, 9]. On one hand, timeout-based aggregation methods collect and propagate packets which were generated in a configured time period. On the other hand, quorum-based methods aggregate packets until a fixed number of packets are merged. This kind of aggregation assures the reduction of bandwidth requirements but does not limit packet transmission delay, contrary to the timeout-based approach.

2.3 Architectural and Middleware Support for MMOGs

Next we will characterize the different network supports used in MMOGs and how they can influence key aspects of the system.

2.3.1 Network Architectures

In this section we explore existing architectural solutions used in the gaming scenario, some commercially adopted and others academically used.

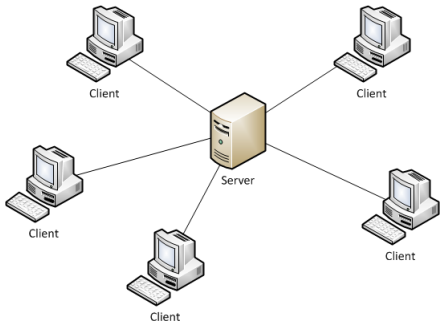


Figure 2.6: Client-Server Architecture

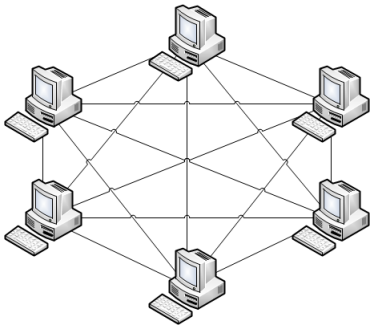


Figure 2.7: Peer-to-Peer Architecture

Client-Server

In a CS architecture[27], every client communicates with a game server responsible for maintaining the current game state (Fig. 2.6). Each player sends its updates to the server which receives this information, resolves any possible consistency conflicts between concurrent updates and advances the game simulation, sending the resulting state back to the players.

This model became popular thanks to its simplicity of implementation, since there is only one connection per client, and state maintenance, since only the game engine in the server side generates game results and the results are globally identical among clients, making cheating a harder task. The CS model is especially advantageous to the game providers, who can easily control game subscriptions, spread game engine updates and monitor game activity.

There are some problems with the CS architecture however. First, the server's scalability is limited to its processing power and bandwidth capacity, thus it can only accept a small number of clients. Second, the server is a single point of failure, able to compromise the entire system if it becomes off-line. Moreover, clients may hold their input procedure until the response from the server arrives and this can possibly prolong the network latency between the server and clients. To increase the scalability of this solution and mitigate the bandwidth requirements many systems employ the distributed client-server model, also called the proxy-server model[20].

Peer-to-Peer

In the P2P gaming model[13, 1], each player (peer) is responsible for executing its own game simulation, eliminating the need for a centralized server. To send updates to the system, a peer has to broadcast them to the system hoping to reach every other peer (Fig. 2.7). The lack of a centralized

server brings several advantages to the P2P architecture. First, it eliminates the single point of failure. This model replicates the game state through all peers, being able to overcome the disconnection of one or more peers. Second, it does not have a server bottleneck. Although the P2P architecture has a higher bandwidth requirement at the players than de CS architecture, that requirement increases linearly with the number of players. Also, there is no delay related with the exchange of updates with the server in the process of synchronization, reducing latency levels. Finally, the P2P model provides increasing scalability with the addition of new peers by assimilating their computational resources to the system.

However, the P2P model is not as popular as the CS in the gaming industry[27], mostly because they are complex of implement and configure due to the communication protocol between peers and the distributed agreement protocol needed to detect inconsistencies among updates. Many P2P solutions [12, 13] use Pastry to map the participating nodes and the application objects, supporting object insertion and lookup, with a Scribe infrastructure to multicast messages. Another problem associated to the P2P model is the need to assign special responsibilities to peer nodes which can compromise the game play as each peer presents different computational and communication resources.

There are solutions, such as Peer-to-Peer with Central Arbiter[40], that merge the P2P and CS solutions (Fig. 2.8). In PP-CA, players exchange updates as they would in the P2P model. Moreover, each player sends his updates to a Central Arbiter. The role of the CA is the same as the server in the CS model but it only sends a correction update, instead of the latest game state to all clients when there are inconsistencies. If inconsistencies are rare events, the bandwidth requirements will be lower than in the CS model but higher than in the P2P model.

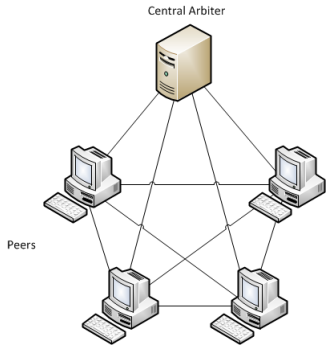


Figure 2.8: Peer-to-Peer with Central Arbitrer Architecture

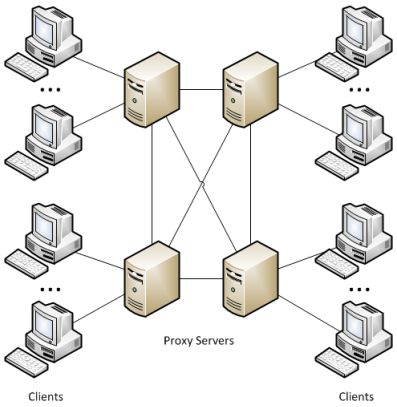


Figure 2.9: Proxy-Server Architecture

Proxy-Server

The Proxy-Server model[20] or distributed Client-Server model (Fig. 2.9) was designed in an attempt to deal with the scalability restrictions of the centralized CS model by using several proxy servers. This model proposes that a client connects to the proxy server which is geographically closer to him and all servers are connected through a P2P structure.

When to submit an update, a client sends it to a proxy server, which sends back an acknowledgement, if the update is validated successfully, and forwards it to all other participant servers. Each proxy will then update his game state and, finally, when all proxies are updated, send the update to all his *avatars*. Although this model offers greater scalability than the CS model and requires less bandwidth from clients, it presents a larger delay associated to the update process since it requires synchronization between proxies.

	Topologies			
	CS	P2P	PP-CA	PS
Network Nodes	Server/Client	Peer	Peer/CA	Server/Client
Number of Servers	1	0	1	any
Number of Clients	limited	any	limited	any
Disposition	centralized	distributed	both	distributed
Point of Failure	single	multiple	both	multiple
Complexity	simple	complex	complex	complex
Consistency Authority	yes	no	yes	yes
Scalability	limited	yes	limited	yes
Update Dissemination	state	any	any/operation	any
Client CPU Load	low	high	high	low
Client BW Load	low	high	high	low
Client Communication	unicast	multicast	multicast	unicast
Server CPU Load	high	-	high	high
Server BW Load	high	-	very low	very high
Server Communication	multicast	-	unicast	multicast
Commercial Use	widely	little	no	no

Table 2.2: Network Architecture Comparison

Independent from a specific topology, there are other architectural features that characterize a system, among them update propagation techniques. There are two major types of update propagation: **unicast** and **multicast**.

Unicast is a transmission technique that sends packets from one host to another at a time. Based on point to point communication, in unicast each process waits for messages through a port (or ports) and unique port numbers are assigned to the process for identification purposes. The most advantageous characteristics of unicasting are its simplicity of implementation and broad acceptance in network devices. However, compared to multicast, it requires more bandwidth to propagate packets to multiple nodes because each message, with the same content, has to be copied and transmitted to multiple destinies.

Multicast techniques[32], on the other hand, can transmit packets to multicast groups at the cost of sending one packet. To receive packets from the multicast group, the node has to subscribe (or join) it. The challenge in the design of a multicast-based application is how to categorize all transmitted information into multicast groups. Another problem related to multicast is that not every router allows this technique as they need to store the state of each multicast group.

2.3.2 Real-time Strategy Games

RTS games, first made popular by Dune 2[3] released in 1992, are generally characterized by three types of action: resource collection, unit construction, and battles between enemy units. In this class of games, the player assumes the role of a god like supreme commander, which commands different kinds of units presenting different capabilities through the map with the goal of defeating enemy troops and gaining control over the scenario. Today RTS games reached a huge popularity thanks to sagas like Warcraft[4], Command and Conquer[5] and Age of Empires[6], each one focusing on different realities: fantasy worlds, armed combat and human evolution. There are several design choices to attend when developing a

[3]http://dune2k.com/duniverse/dune2
[4]http://www.blizzard.com/war3
[5]http://www.commandandconquer.com
[6]http://www.microsoft.com/games/empires/

strategy game, such as:

- ***real-time vs turn-based.*** One of the most decisive choices when developing a strategy game is to chose between real-time strategy and turn-based strategy. On one hand, turn-based strategy (TBS) ports the format of strategy boardgames, such as Risk[7], to the virtual environment. Each player can only make moves at his turn, having to wait for all remaining users to play before playing again, which can take a long time. Additionally, player's moves during his turn are limited. On the other hand, in real-time strategy (RTS), players can perform actions simultaneously and without limitations, bringing dynamism to the game. Although RTS is more realistic and appealing to most users, it needs consistency enforcing mechanisms not necessary in the TBS approach.

- ***map partitioning.*** Over the years, many map partitioning techniques have been employed. On Warcraft 2,[8] developers opted for partitioning the global map into square tiles, allowing objects to move to any adjacent square. TBS games, such as Civilization IV,[9] usually choose to partition the map using hexagon tiles. By partitioning the map, developers can easily calculate the distance between objects, ranges of attack and collisions between objects (when they occupy the same tile). However, this approach is not realistic. Nowadays, most RTS games do not partition the map, making object movements much smoother and realistic. Nevertheless, game engines have to detect collisions between objects.

- ***multiplayer partitioning.*** In order to avoid the scalability problems associated with centralized multiplayer management, many systems opt for distributed solution, partitioning the global map for all available servers. In the partitioning process, each server is assigned an area of the map with sizes depending on the network and communication resources available, guaranteeing the load balancing of the system. This way, servers with stronger resources are assigned with bigger areas and servers with weaker resources are assigned to smaller ones. Map partitioning is usually made into rectangular or hexagonal areas, however it disregards objects disposition. Other approaches try to partition the virtual world using computational geometry techniques, such as Voronoi Diagrams[37], that partitions the game space into polygonal regions based on locality properties. Although Especially effective in RTS game where object activity is frequently limited to a particular region, the time complexity associated with Voronoi Diagrams is a major disadvantage.

- ***2D vs 3D.*** Since the computer graphic evolution started, many legacy games, such as Warcraft, have developed titles that explored the 3D capabilities. While games like Warcraft 2 tried to explore a sense of 3D by picturing building and characters in perspective, the animation of objects through the map, done using sprite sheets, and limited user view of the scenario made clear it is a 2D game. The sequel, Warcraft 3, is already a full 3D game, where every entity is a 3D object and also the map. One of the greatest differences between Warcraft 2 and 3 is on the camera control. In Warcraft 3 the player have a wide spectrum of possible camera movements, providing great dynamic to the game.

  Next. We establish a comparison between two popular strategy games: StarCraft and Civilization V, presenting the design choices of each one.

[7] http://www.hasbro.com/risk/
[8] http://eu.blizzard.com/en-gb/games/legacy
[9] http://www.2kgames.com/civ4/

- ***StarCraft.*** Released in 1998, StartCraft[10] is a space themed 2D RTS in which the results of the victory or defeat on the game will be decided by strategies of each player. The game map is characterized for being partitioned into square tiles. The main attraction of StarCraft is the network play. StarCraft presents its own dedicated game server system, Battle.Net, which works as a game lobby system that provides game sessions for players all around the world. Although players have to connect to Battle.Net to find a game session, it works only as an intermediate system, establishing a client-server connection between participants where one acts as the server. In essence it acts as a broker or a game server.

- ***Civilization V.*** Civilization V[11] is a 3D turn-based strategy game released in 2010. In this game, the player assumes the role of a grand commander and has to lead his nation for glory by conquering every other nation or surviving until the year 2050. As most TBS games, the game map is partitioned into hexagonal tiles and entities only exist in units, each with different combat strength depending on the number of battles it participated and vulnerabilities. Network playing is only possible using the STEAM[12] system dedicated server system.

Figure 2.10: Comparison between StarCraft and Civilization V

2.4 Global Analysis

After exploring the state-of-the-art in the fields of consistency management, interest managing and architectural support, it is possible to conclude that consistency maintenance is achievable at low bandwidth costs by dynamically identifying the different consistency requirements objects present at runtime.

This way, a model like VFC, which combines IM techniques with continuous consistency management, is a suitable choice for reducing bandwidth requirements and server CPU load by distributing it through the clients.

[10] http://us.blizzard.com/pt-br/games/sc
[11] http://www.civilization5.com
[12] http://store.steampowered.com

Chapter 3

Architecture

To prevent divergences between each player's local game state, potentially compromising gameplay, the majority of commercial RTS titles employ a *total consistency model* where, independently of the network architecture employed, each game state update is propagated to all participants. To achieve total consistency, considering a client-server solution, each player sends his updates to the server, responsible for maintaining the most accurate game state, solving conflicts between concurrent updates and disseminating the latest game state to the remaining clients. Although this solution presents favorable results when applied to small systems, the same does not apply on systems with a large number of clients and a high update creation rate, since the server becomes overloaded with client requests and incapable to respond to them timely. This situation can be overcome by increasing the server's CPU and network capabilities. However, system scalability will remain limited to the server's capacities, constituting a *bootleneck* to the overall system performance.

One possible approach to deal with these scalability limitations is the addition of more servers. This will contribute to a more scalable system since client connections can be divided by the servers, resulting on a load balancing improvement. However, this solution greatly increases the consistency management complexity, as every server is eligible to receive updates, requiring constant communication between the servers in order to calculate the latest game state. Another approach to increase scalability stands on delaying update dispatch on the server. To perform it, the server must be able to identify which updates are most relevant, on a client by client basis, in order to dispatch them first, delaying the less relevant ones. This may also result on the merging of less relevant updates as new updates may nullify or supersede old ones. The outcome of this approach is the reduction on network resources consumption as well as prevention against the server being flooded with incoming requests, unable to respond within acceptable time.

In this chapter we start by examining the VFC model and how it can be adapted to the RTS environment (section 3.1). Next, we present the architecture of the VFC-RTS middleware and how it can be applied to a RTS game (section 3.2). Finally, in Section 3.3.2, we explain in detail the consistency enforcement on the VFC-RTS middleware.

3.1 Applying VFC to Real-Time Strategy games

As described on sections 2.1.3, VFC is a continuous consistency model that employs locality-awareness techniques in order to maintain the consistency requirements of each client's local replica, without propagating the full game state. Previous adaptations of the model for FPS[19] and Arcade[29, 15] games consider the user's location as being the position of the character controlled by him, as there is only one. However, that is not the case on a RTS game, owing to the fact that a single user is responsible for multiple entities.

In this section we will present how VFC can be adapted to fit the consistency requirements of the RTS environment, emphasizing how concepts such as *pivot points* and *consistency zones* are adjusted to this new reality.

The VFC model represents the virtual world as a N-dimensional space populated by game objects which can be player's units or structures, bonus items or computer controlled entities. The state of the virtual world is defined as the aggregation of the states of all active game objects. Each player keeps a local *replica* of the game state, however, only the server holds the most accurate global state. Responsible for update reception and conflict resolution, the server is the key enforcer of the VFC model, sending object states according to the consistency requirements established on a player by player basis.

Figure 3.1: Example of *Pivot* application on a RPG game

On VFC, the user's AoI is defined using the concepts of *pivot* and *consistency zone*. A pivot point represents the position around which the local replica presents strong consistency requirements, weakening as the distance to the pivot increases. For example, in a RPG game, a pivot point would be established at the player's avatar position since the player needs continuous intel about occurrences on his surroundings (Fig. 3.1). One possible adaptation of this concept for RTS is establishing a *pivot* on each object controlled by the player (Fig. 3.2). This way we assure every event regarding the player's entities is reported to him, an important requisite when delineating strategies. However, this solution does not fill all the consistency requirements of real-time strategy since the player may not be aware of events taking place on the area captured by the camera if there are no player units present. Also, this solution may require excessive calculations as we will describe further on this document.

Figure 3.2: Example of *Pivot* application on a RTS game

To enforce the boundaries between consistency levels, the VFC model introduces the concept of consistency zones. Consistency zones are concentric, ring-shaped areas described around a pivot point, presenting increasing radius and divergence degrees. In practical terms, the consistency zones are used to quantify an object's importance. This way, objects positioned in zones closer to the pivot present more relevance and as such are refreshed more often than objects further apart from the pivot point. Since this work targets RTS games, which nowadays are most commonly 3-dimensional, we opted to carry over the original consistency zones design to 3D, describing consistency zones as concentric spheres (or half spheres since most game objects are standing on the terrain).

The proposed adaptation of the VFC model is composed of two parts: adaptation of the player's area-of-interest to the map area captured by the camera and aggregation of entities into units according to the distance among them.

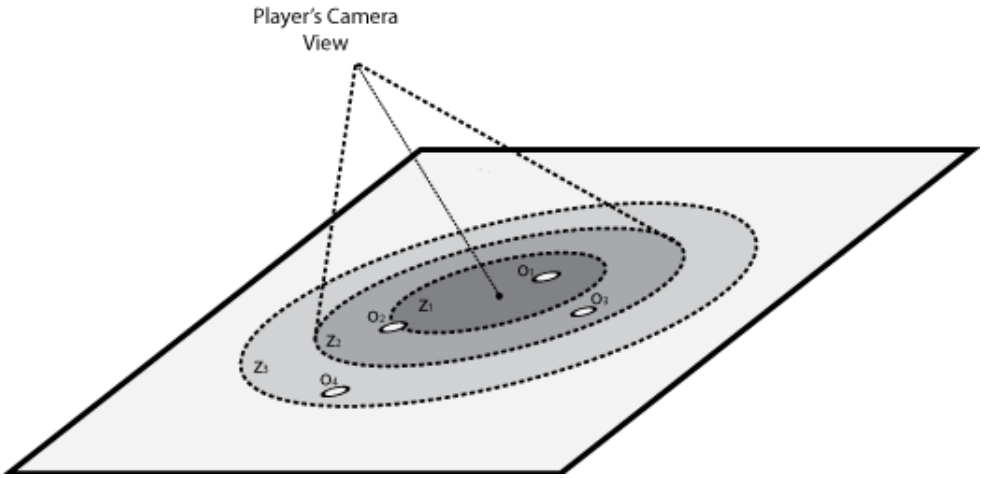


Figure 3.3: Adaptation of player's AoI to his camera view

In Fig. 3.3 we present an illustration of the proposed solution. By assigning the pivot point to the center of the player's camera view, we guarantee that every object within the player's area of visibility presents an acceptable divergence degree for the correct execution of the game. Furthermore, the flexibility

of VFC and its consistency zones adds dynamism to the solution. By guaranteeing bounded consistency of the objects in the outskirts of the player's view, this solution makes it possible for the user to change the camera's position without compromising the user's game experience. When this event takes place, the VFC's configurations take the new location into account and dynamically assure the consistency of the objects presented in the new AoI.

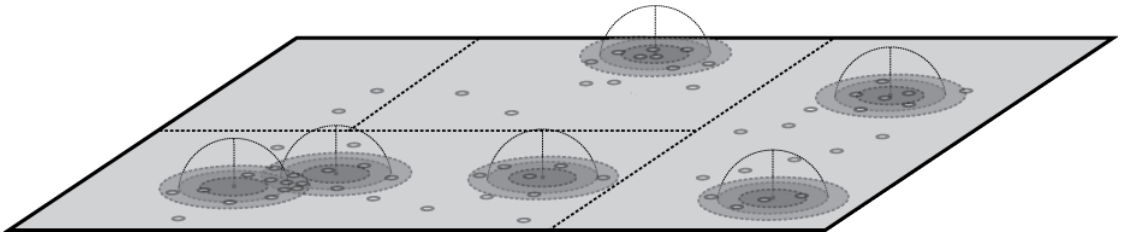


Figure 3.4: VFC-RTS multiplayer scenario

We believe this pivot point attribution best serves the consistency requirements of real-time strategy since the majority of strategy commands and decisive battle events concerning the player take place inside his camera spectrum. This way we assure the player's vision of the game is always as accurate as the server's, providing him all the information required for future commands. Although the presented adaptation of VFC covers most of the player's area-of-interest, there can also be events affecting the user not followed by the camera. An example of this situation is the player forming an unit and ordering it to attack an enemy structure while the camera is set on the player's headquarters. To cover this situation, we propose next an additional feature to the VFC-RTS solution - the concept of entity aggregation into units (Fig. 3.5).

A unit is a group of entities controlled by the same player performing the same set of actions within a limited time period. Especially useful for position changing updates, if the system detects entities within a fixed distance threshold going on the same direction, it establishes a pivot point in the center of the unit formation and consistency zones. Unit formation can also be triggered by the user making use of game controls, enabling a more precise entity control. Not only this process allows following entities outside the camera span without compromising the consistency of the system, it also reduces the processing and communication load since update messages regarding entities of the group can be aggregated.

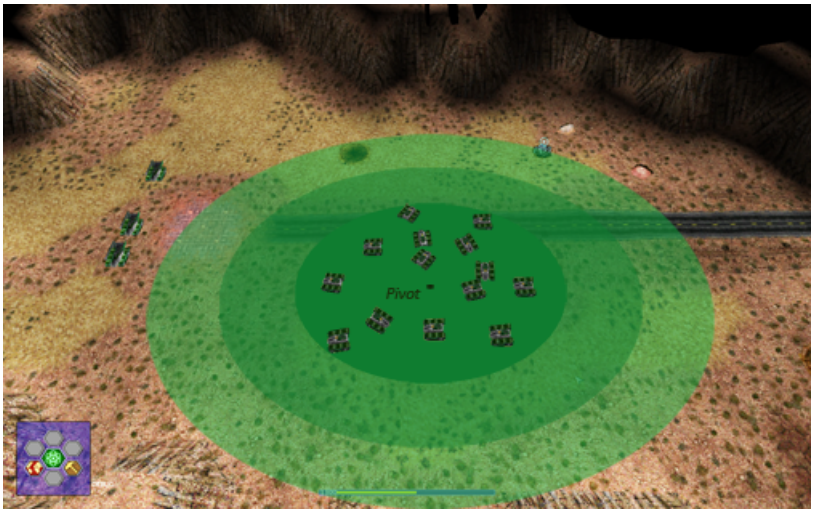


Figure 3.5: Example of entity aggregation - Unit

3.2 Middleware Architecture

On a RTS game, as on other game categories, when playing in multiplayer mode, all player clients contribute to the game state by operating on the scenario, interacting with game objects as well as other players. Although they are given the illusion of manipulating a single-copy state, each one is interacting with a local *replica*, propagating the game state updates to the remaining players.

In this section, we will present in detail how a system using VFC-RTS manages game state consistency by explaining the message exchange protocols and the different components involved in game state updating. We start by describing, in section 3.2.1, the *Network Architecture* of the system followed, in section 3.2.2, by a detailed description of the *Software Architecture*.

3.2.1 Network Architecture and Protocols

Following the architecture adopted by the original VFC model, the VFC-RTS distributed framework is composed of two types of network nodes: **client** nodes and a single **server** node.

Running an instance of the game, clients are accountable for any changes to the game state and submission of local replica modifications to the server node. In addition, since VFC integrates the concept of *locality awareness*, the client is responsible for proactively informing the server about changes regarding pivot position. For both purposes, the client employs two message types VFC-RTS specific, presented further in this document. The server node is responsible for game state update reception, applying newly received updates to the global game state, solving possible conflicts between concurrent client updates and propagating the most recent game state according to each player's consistency requirements.

The network node distribution is illustrated on Fig. 3.6.

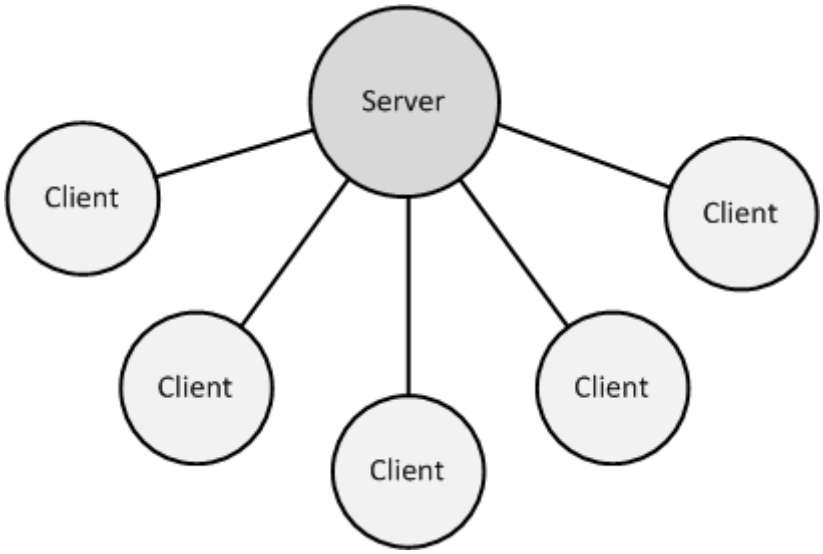


Figure 3.6: Network node distribution on VFC-RTS

Although a client-server topology is a less scalable solution when compared to a totally distributed architecture such as peer-to-peer, it provides several advantages. First, since there is a single point of control for game state updating, maintaining the consistency of the system becomes a much simpler

task. This way, the need for inter client synchronization whenever a conflict happens is discarded, which may result on message exchange optimization. Secondly, all VFC reasoning is confined to the server, resulting on a highly simplified process of integration of the middleware into new games. Also, since the server is the model enforcer, only this node needs to be aware of player's pivot position updates, once more reducing the number of exchanged control messages. Finally, the client-server topology is the easiest and most intuitive architecture to implement, granting the additional bonus of always having a node presenting the full game state, beneficial for system monitor purposes, security control and cheating avoidance.

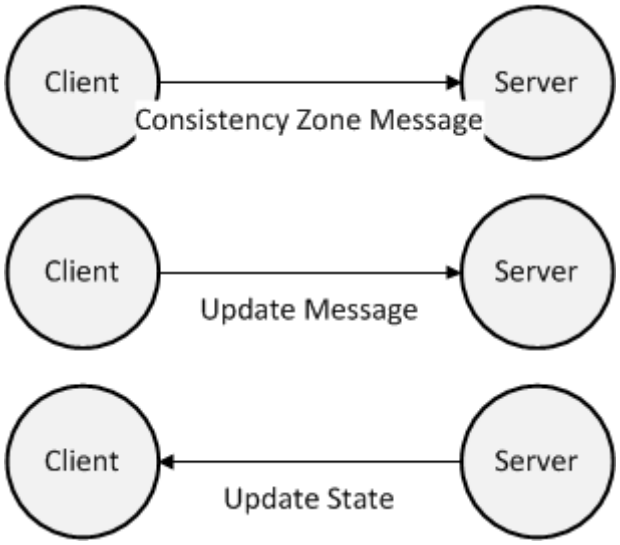


Figure 3.7: Messages exchanged between the Clients and Server

Client and Server Messages

The VFC-RTS model presents three message types to be exchanged between client and server (Fig. 3.7):

- ***Consistency Zone (CZ) Message***. Used to inform the server about *pivot* position updates, these control messages present a very simple structure with the pivot's id, the new coordinates (represented using two or three dimensions depending of the system) and the radius of the maximum consistency zone, the area immediately surrounding the pivot (the remaining consistency zones are calculated based on this radius). Consistency zone messages represent any kind of pivot, regardless if it being a unit pivot or the camera pivot, and are dispatched whenever the distance between the last sent and new pivot position exceeds the configured threshold value.

- ***Client Update Message***. This message carries information about changes on the client's local replica to the server. The representation of this message is dependent of the target entity type and operation (e.g. moving a droid, attacking a structure), and suitable for state-transfer as well as operation-transfer systems since client-side update integration is independent from VFC logic, further contributing for the abstraction of the system.

- ***State Update Message***. Used exclusively to send game object states to clients, this message is dispatched from the server according to each player's VFC enforced limits. Although this message is used to update local replicas and has no standard representation (as the previous message), it is important to notice it follows a state-transfer policy. This happens to guarantee that if a replicated

object does not receive updates for a long time, in a single State Update Message, the replica becomes synchronized with the primary object, instead of receiving all the operations that took place since the last update.

Although the presented message types represent the core of the VFC-RTS model enforcement, they do not preclude the use of some game oriented message types. Examples are game specific control messages for game time synchronization and for creation and destruction of game objects. Since these messages may have a direct impact on game and VFC logic, they are not subject to delayed dispatch imposed by the VFC algorithm, being immediately redirected once arrived to the server.

3.2.2 Software Architecture

Client

The Client node corresponds to the adapted game application run by the user. Asynchronously modifying its local replica, the client is given the illusion of interacting directly with the full game state. Whenever the client modifies the local game state, an update is sent to the server node. Thus, the client node monitors user activities and listens to incoming server update messages. As soon as updates arrive, they are immediately applied to the local replica. The client node architecture is presented on Fig. 3.8.

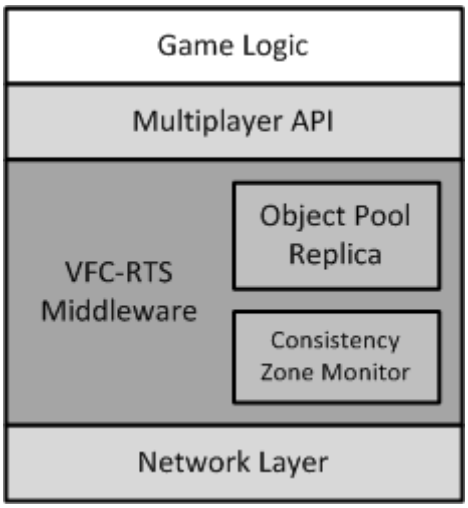


Figure 3.8: Client node Layered Architecture

Analyzing the layer configuration of the client node, a prominent feature of the VFC adaptation is the abstraction between the game and VFC logic. As far as a client node knows, all game states updates are locally received, resulting on an updated vision of the entire map extent. This feature results on very little or no changes to the target client application when adapting to the VFC model. Additional to the game logic layer, a VFC-RTS client presents three layers:

- **Multiplayer API Layer.** This layer is responsible for the interface between the VFC model and the client application. Through this layer, the application injects game updates from other clients, and dispatches local changes through the network. The importance of this layer resides on the abstraction it provides to the system. By employing a multiplayer API, the integration of the VFC model is smoother and easier to achieve.

- **VFC-RTS Middleware Layer.** The VFC-RTS Layer is the core enforcer of the client-side VFC model. The main components of this layer are the **Object Pool** module and the **Consistency Zone Monitor** module. The Object Pool maintains a local replica of the collection of all game objects. Objects located within areas marked as areas-of-interest of the client present stronger consistency levels than objects outside them. The Consistency Zone Monitor assures the correct function of the VFC model since it is responsible for the detection of changes to the consistency zones configurations and forwarding them to the server. Hence, whenever the player changes his camera view position (above the configured threshold value), or changes the location of a group or a group's configuration (i.e. number of entities in the group, max consistency radius), a new consistency zone message is dispatched to the server.

- **Network Layer.** Responsible for the actual client-server communication, the Network Layer is where all game and VFC information enters and leaves the system. It is important to notice this layer does not obey to a specific network protocol, giving developers the freedom to employ the protocol most suitable to the target system.

Server

The Server is the central node in the system. Every message exchanged in the network is either sent or received by this node. It accumulates client management and update propagation responsibilities. Although the server immediately applies client's game state updates according to the arrival order, in a VFC powered system, it is up to the server to reason when to send object state updates in accordance to each user's AoI. The server node architecture is presented on Fig. 3.9.

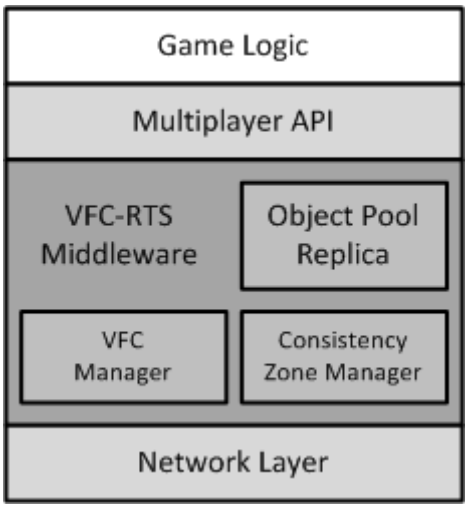


Figure 3.9: Server node Layered Architecture

Since VFC-RTS is the one layer with major differences comparing to the client architecture, we will continue this section by presenting the two most important modules in this layer: the *Consistency Zone Manager* and the *VFC Manager*.

Consistency Zone Manager. The Consistency Zone Manager holds in the server all the information regarding every player's consistency zones, a critical element in the VFC decision making process (Fig. 3.10). With this knowledge, the VFC model can accurately reason when to dispatch object state updates

to players clients, demanding for these real-time updates about any changes to the player's consistency zones. Thus, if the difference between a pivot position and the last position sent to the server rises above a configurable threshold, a new location message is sent to the server.

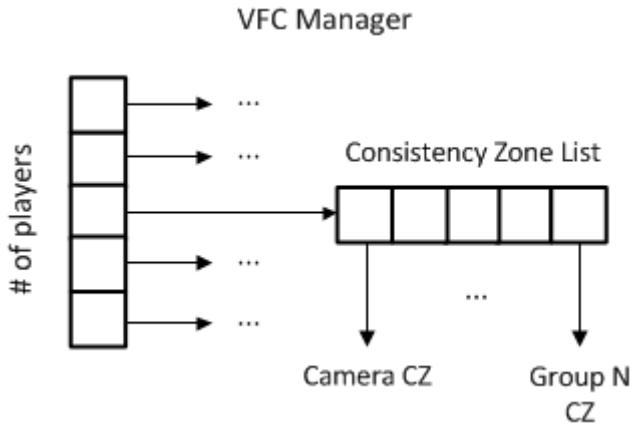


Figure 3.10: Consistency Zone Management

VFC Manager. The VFC Manager is the central component of the Server node and the core component for VFC enforcement. Its main responsibility is the enforcement of the VFC algorithm, in conformity with each client's consistency requirements. The algorithm is applied to every object in the game in order to detect eligible updates for dispatch. This process is repeated for each client since each one have its own AoI. In addition to this task, the VFC Manager is also responsible for tagging which objects have been subject to updates and, if that is the case, to which client's have the new object state been sent (Fig. 3.11). To do this, the VFC Manager monitors every incoming client update to identify which objects are updated. For each updated object, the server keeps a record of what players have already received the new state so that every client may, sooner or later, be aware of this game state modification.

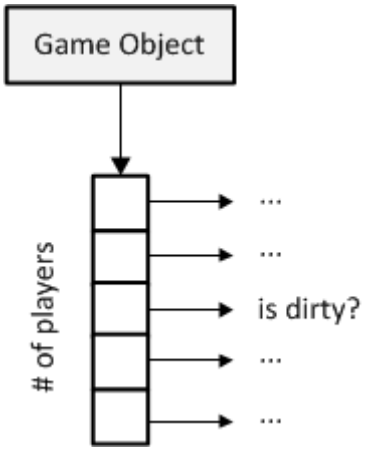


Figure 3.11: Game Object Modification Management

Client-Server Message Flow

In Fig. 3.12 we illustrate the message flow between the clients and the server. In this representation we can easily identify the three update processes that characterize our VFC architecture: *Client Updates*,

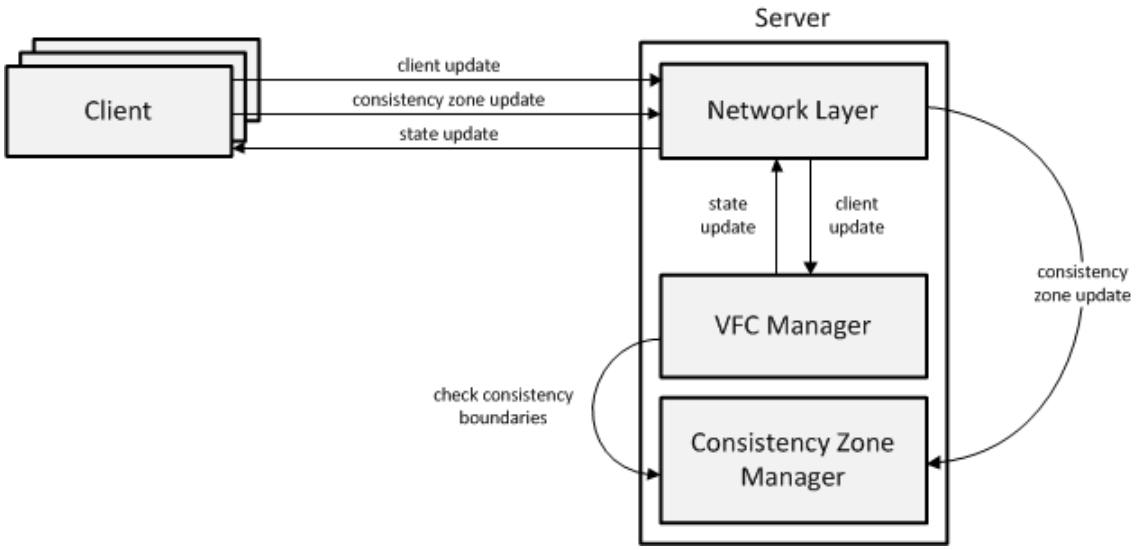


Figure 3.12: Client-Server Message Flow

Consistency Zone Updates and *State Updates*.

In a client update, the client asynchronously sends to the server the information regarding modifications to the local replica so they may be applied to the primary game state and, consequently, to the remaining clients game state . This updates have to be sent immediately after the replica update to maintain the server's game simulation complete consistency, a feature required to guarantee the model's correct behavior, as well as game rules. When arriving to the server, the update is interpreted by the VFC Manager in order to determine which objects are targeted for modification and then applied to the primary game state.

Triggered whenever a client changes the configuration of a consistency zone, Consistency Zone Updates provide the server all modifications regarding the player's AoI. Once in the server, this information is stored in the Consistency Zone Manager so it can be looked up during the VFC reasoning process.

Considering object o and client c, whenever the VFC Manager detects that o is tagged as dirty and does not respect the consistency boundaries set for c, the server sends c a State Update Message, containing the o's most consistent state (the primary state). Furthermore, the server flags a state update regarding o was sent to c. This will prevent the server from repeatedly sending clients an object's state before it is once again significantly modified (taking into account the consistency zones).

3.3 VFC Enforcement

The server node is accountable for enforcing the VFC model. This particular task is assigned to the VFC Manager module, which monitors the fulfillment of every client's consistency requirements during a game session. As soon as an object reaches a client's stablished VFC limits, the state of that object is immediately propagated.

In this section we start by presenting the three dimensions used to express the VFC Consistency Vector limits (section 3.3.1), followed by the explanation of how they are employed for Consistency Enforcement (section 3.3.2).

3.3.1 Vector-Field Consistency Limits

To formalize the consistency degree of a zone, VFC employes a 3-dimensional consistency vector:

Time (θ): The maximum time (in seconds) a local object can stay without being refreshed with its primary replica's latest value. With this metric, VFC guarantees that an object within a consistency zone has to be refreshed within intervals of θ seconds. In the practical sense, this dimension expresses the "freshness" of an object.

Sequence (σ): The maximum number of updates to the primary object state not applied to the local object (missing updates). With this metric, VFC guarantees that an object within a consistency zone has to be refreshed within intervals of σ lost updates.

Value (ν): The maximum divergence between the contents an object's local state and its primary state. Value is an application-dependent metric and by that reason must be calculated using a function implemented by the application developers.

For the purposes of our system, ν represents the maximum distance between an object's primary replica position and local replica position, i.e., a measure of the error associated with the perceived location of an object (in the local replica) and its actual location (in the primary replica). We understand that, the bigger the consistency requirements, the smaller must the distance threshold be for replica synchronization. This feature will also guarantee the model's correct behavior in a case where an object o, presenting a great distance from any of a player p's pivots, takes a move command that will put o inside p's AoI. Although not covered on the other consistency vector dimensions, the maximum distance between an object's replicas position may be a key factor for consistency maintenance in a RTS game.

3.3.2 Consistency Enforcement

The VFC model is enforced using a two part algorithm. The first part consists in keeping track of all updates to the objects of the system, maintaining the information required to enforce client's consistency. The second part of the algorithm consists in updating the client's game view according to his VFC settings.

To enforce the VFC model the following data structures are required:

- ***Dirty List.*** A 2-dimensional list with number of players by number of objects dimensions (Fig. 3.11). The Dirty List stores each object's information regarding changes to the object state and which clients are already aware of them.
- ***Current State Vector.*** The Current State Vector keeps information about the state of all clients for lookup during the VFC reasoning process. The vector stores the last time o's primary state was sent to c, the number of updates since the last time that o's primary and local replicas synced, and the Value o presented the last time its state was sent to c.
- ***Consistency Zone List.*** A 2-dimensional list with number of players by number of consistency zones dimensions (Fig. 3.10). This list stores all the information regarding each client's consistency zones and is used for look up during the VFC reasoning process.

```
NEW_UPDATE(OBJ)
 1. for each c in CLIENTS
 2.    vector := currentStateVector(OBJ, c);
 3.    if vector.time == -1 then
 4.    vector.time := current_time;
 5.    vector.sequence++;
 6.    if vector.value == 0 then
 7.       vector.value += DISTANCE(OBJ.current_position, OBJ.new_position);
 8.    DIRTY(OBJ)
```

Figure 3.13: Handling new client updates

```
VFC_MONITOR()
 1. for each o in OBJECTS
 2.    for each c in CLIENTS
 3.       vector := getMaxConsistencyValues(o, c);
 4.       if isDIRTY(o) == true then
 5.          if !CHECK_VFC_CONDITIONS(o, c) then
 6.             sendStateUpdate(o, c);
 7.             updateCurrentStateVector(o, c);
```

Figure 3.14: Pseudo-code of VFC Monitor

New Update Received

When a new update is received it is necessary to tag the referring object as dirty to all clients and update the object's Current State Vector (Fig. 3.13). This operation consists on: checking if the time field already has a value and if not give it the current time; increasing the number of missing updates to the object by one; checking if the distance between the object 's primary and local replicas is null and if so give it the current distance, adding the current distance to the present value otherwise. Finally, the dirty flag is activated to all clients.

VFC Monitor

VFC updates a client's local view by sending it object state updates. Since objects and consistency zones are in constant mutation, it is necessary for VFC to monitor object activity during the game session. This task is assigned to the VFC Monitor function (Fig. 3.14) which dispatches updates to each client, based on the data gathered during the New Update Received process.

The algorithm works as following (considering a pivot p): identifying the objects within p's consistency zones; then, for each object, identifying in which consistency zone it is located; verifying if the object breaches the consistency zone limits (if an object is inside more than one, consider the one with stricter consistency requirements) and if so, a State Update Message is sent. After, sending the newest object state to the client, the server has to update the Current State Values for that client by setting θ to -1, σ to 0, ν to 0 and clearing the dirty bit.

```
VFC_UNIT_MONITOR()
 1. for each u in UNITS
 2.    for each o in OBJECTS
 3.       if positionChanged(o) == true then
 4.          calcPivot(u);
 5.          calcRadius(u);
 7.          sendCZUpdate(u);
```

Figure 3.15: Pseudo-code of VFC Unit Monitor

VFC Unit Monitor

Since an unit's consistency zone configuration may change as a result of the operations given to the droids in it, it is necessary to inform the server about these modifications. To keep track of any changes to the unit's consistency zone, we have developed a VFC Unit Monitor (Fig. 3.15). The unit monitor uses a very simple algorithm: if an object o belonging to a unit u changes its position in the map, u's geometric center is updated, as well as its radius, followed by the dispatch of u's new configuration to the server.

Although in this example we consider the VFC Unit Monitor running in client-side, it can also run on the server, requiring however the server to be notified about any droid addition or removal to the unit, which may result on additional network traffic.

Chapter 4

Implementation

In this chapter we describe some details about the implementation of our solution. We start by presenting the target application, explaining the reasons that lead us to choose it. Next, we explore how communication is handled before and after the application of the VFC-RTS middleware, underlining the data structures created or adapted for this purpose. Finally, we analyze how the VFC concepts were integrated in the application and how they the affect user interface.

4.1 Warzone 2100

In this work, we applied our solution to a RTS game. The game chosen for this purpose was *WarZone2100*[1], an open source, futuristic, 3-dimensional war game. In this game, the player (referred to as "The Commander") controls a great variety of entities (referred as "Droids") which can be enhanced using the massive research tree available. Compared to other RTS games, Warzone 2100 has a greater focus on artillery, radar, and counter-battery technologies.

The main reasons that led to choose WarZone2100 as the target game for the implementation of the solution were:

- Mostly written in C++ and compiled for the main Operating Systems (Linux, Windows, OS X).
- Wide library of maps with different dimensions, providing the possibility on creating new maps using the map editor.
- Commercial game originally developed by Pumpkin Studios and published by Eidos Interactive, now developed by the Warzone 2100 Project with open-source code available on [2].
- Still in development, with a large community and forum for developers.
- Full 3D environment, perfect to test the application of the VFC algorithm to a 3D environment.
- Already presents a Multiplayer option up to 8 players.

[1] http://wz2100.net
[2] http://developer.wz2100.net

4.2 Network Communication

Although the network configuration used on Warzone 2100 multiplayer has morphed through the various versions of the game, for this work, we opted to apply the middleware to the game's latest version. In this version, a player (host) starts a multiplayer session and waits for other players to join in before starting the actual game. The host acts as the multiplayer game server, responsible for accepting player connections prior to the multiplayer game start and redirecting incoming player messages to all the remaining players. It is important to mention that after the host starts the game no other players can join in.

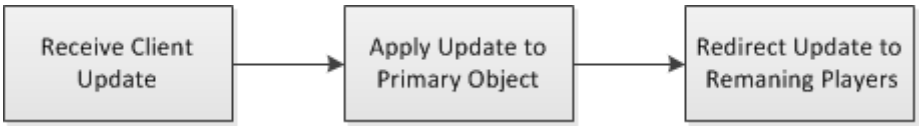


Figure 4.1: Warzone 2100 Update Process

Once the game starts, every player keeps locally the full game state. Consistency is maintained by broadcasting every player's operation to all other player's so that they may apply them locally (Fig. 4.1). To assure operations are synchronously applied in each participant's local view, control messages are dispatched ten times per second. Containing each player's game time, latency and summary of the entire game state, these messages are used for: (a) detecting state divergences between players; (b) signaling the moment when pending operations should be applied to the local game state, guaranteeing that even the player who issued the operation will wait upon the next control message to apply it. Furthermore, if game state divergence between player is detected, the multiplayer game is terminated.

The network communication between players is performed through the NetPlay library. Developed specifically for Warzone 2100, this library employs a simple API that represents messages as a sequence of low-level data types. NetPlay uses two types of messages: game messages, containing information regarding game logic (i.e. player operations), and network messages, used to transmit information of session control nature (i.e. ping). This distinction between message types is made in order to separate game logic from network logic. An interesting feature about this library is that only network messages are actually sent through the network. As all game related information has to be broadcasted, considering a client c and instant t, every c's pending game messages till t are encapsulated into a single network message N_{ct}, which in turn is propagated to all participants.

4.3 Adapting VFC to Warzone 2100

The host is the central point of the presented game network and, for that reason, the most suitable candidate to be the VFC server while the remaining players acts as clients. Since the host is also a participant in the game, it acts as both server and client.

4.3.1 Update Filtering

The VFC model assures the consistency of a system by delaying update propagation according to the divergence boundaries defined for each intervenient. To achieve this, the server must be able to filter

incoming client messages. In WZ2100, this process is assign to NetPlay. Since all messages arrive to the server by the Network Layer, is up to the NetPlay library to block client updates from being immediately redirected to other clients (Fig. 4.2). Although Warzone 2100 presents several types of game messages, DROID_INFO messages are the ones exchanged the most during a game session since they may contain any droid related operation (e.g. attack, move, etc), and for that reason, the only message type considered in this work for VFC delaying.

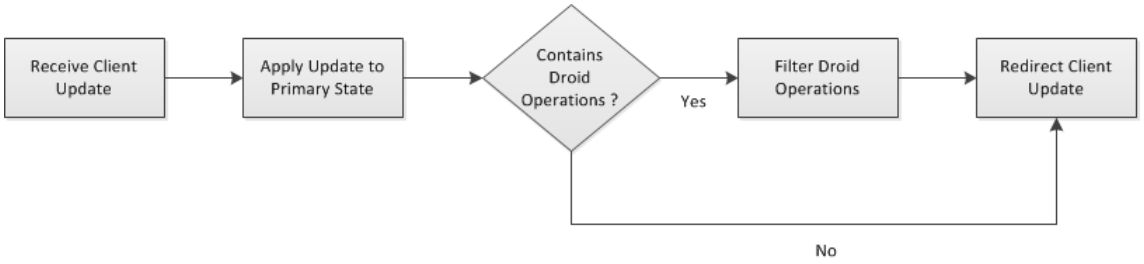


Figure 4.2: Update Receiving Process

The update filtering process consists of: (1) decoding any network messages regarding a client's game operations; (2) applying all client updates to the primary game state; (3) encoding a new network message less the DROID_INFO messages. After the completion of the filtering process, the only nodes aware of any droid related game state modifications are the server and the client who issued them.

4.3.2 Update Insertion and Delivery

Upon the detection of a droid d breaching a client c divergence boundaries, the system must update the c's local state. Since update propagation on Warzone 2100 is made on an operation basis, to enable object state propagation, a new message type DROID_STATE was created. In addition to all the criteria included in a DROID_INFO message (identifying droid d, its orders and targets), but regarding current orders instead of new ones, a DROID_STATE message also contains the current position of d. This way we guarantee that when a client receives a state message, d will automatically be positioned in the same location as the primary replica. Since object creation and destruction messages are immediately forwarded by the server, and client c does not receive information about its own droids, no additional droid information is needed to be sent in order to assure the game's correct behavior.

Once arrived to client c, the DROID_STATE is processed, modifying d's state using a modified Multiplayer API. Although d may present a long distance between the new local state position and the position on the last received update, on the camera view, this is not noticeable by c. Since droid state updates are sent according to the client's consistency requirements, cases of such a divergence between local and primary replicas will only occur on droids far from the client's AoI. Thus, the closer d is from a c's pivot, the smaller is the distance between local and primary state's position.

4.3.3 Consistency Zone Notification

Additional to the adaptations employed to the droid updating system, it was necessary to integrate in Warzone 2100 a consistency zone notification system. Since the VFC reasoning process needs to know, in real-time, about any modifications to the client's consistency zones, it is imperative to inform the server of such occurrences. To do this, we introduce a new message type PLAYER_ZOC.

A PLAYER_ZOC message is a small message containing the consistency zone id, it's current coordinates and the radius of the maximum consistency zone. With this information the server can resize and reposition the consistency zones described around the given coordinates. Since the remaining consistency zones are dimensioned based on the maximum consistency zone radius, no additional information is required.

There are two ways to trigger the PLAYER_ZOC message dispatch: During a move of the camera view, thus moving the camera's pivot point, or during a move of droids inside a droid group. To trigger the dispatch during camera movements, we have incorporated a camera event handler to send a consistency zone notification each time the pivot changes coordinates. For droid groups, we've implemented a group movement monitor that sends the server a notification each time the group's geometric center moves. Although the monitor is currently implemented on the clients, since the server only knows which droids are on its own groups, it can be solely implemented on the server. However, this requires the server to know the composition of every entity group.

4.4 Data Structures

To support the modifications made to Warzone 2100 in order to integrate the VFC model, some data structures had to be created or modified. Following the structure models introduced on section 3.3.2, we will now present how those structures were implemented in the game.

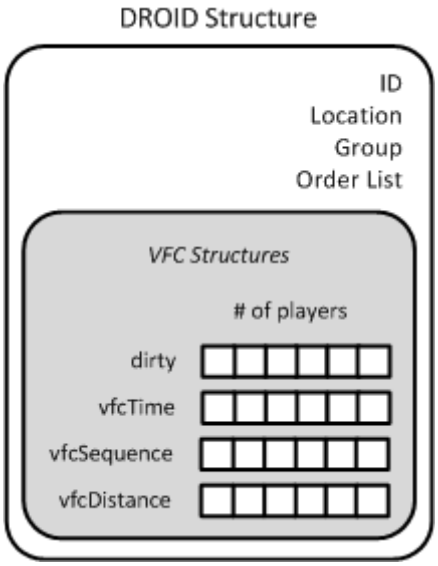


Figure 4.3: Server-Side Droid Structure

To keep track if a droid d has been updated and what clients are already aware of d's last state, the server must keep for each droid a vector with a size equal to the number of players. Similarly to this vector, each droid presents three more vectors: a *vfcTime* vector, to store the last time d's local state was updated; a *vfcSequence* vector, to store how many updates have arrived to the server since the last local state update, and a *vfcDistance* vector that keeps the distance travelled between the local replica position and primary replica position. These vectors are added to the existing DROID structure and modified every time a new client update arrives (Fig. 4.3).

To store information regarding zones of consistency, a new data type ZOC was created. In addition to the information exchanged in the PLAYER_ZOC messages, when describing consistency zones around a droid group, a ZOC structure may contain the list of droids belonging to the group (Fig. 4.4). The list of droids is kept client-side in order to adjust the consistency zone configuration deriving from droid moves. Although each client keeps locally a ZOC vector regarding its consistency zones, since the VFC reasoning process takes place in the server, it stores every client's consistency zones as a matrix (illustrated on Fig. 3.10).

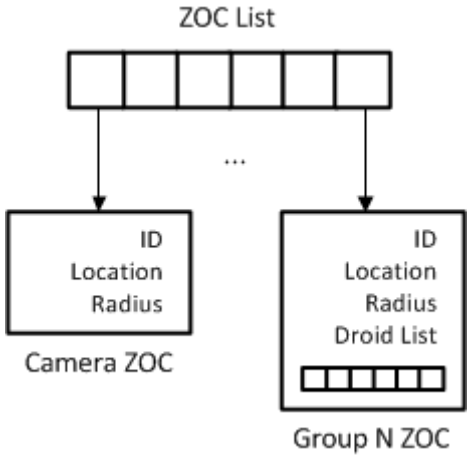


Figure 4.4: Client-Side ZOC List

4.5 Vector-Field Consistency Configuration

In order to adjust the consistency metrics employed by the VFC model to the different consistency zones, all VFC related values are stored in the Warzone 2100 configuration file.

In the configuration file we can find the threshold values for VFC's *time*, *sequence* and *value* from the maximum to the minimum consistency zones and the threshold distance for consistency zone update propagation. Also, since the radius for all consistency zones around a pivot point are calculated based on the radius of the maximum consistency zone, in this file we also keep the coefficients employed to the calculation. This feature is important in order to tune VFC values, optimizing the model application.

4.6 Summary

After completing the implementation of the proposed solution we conclude that:

- The integration of the VFC-RTS framework on Warzone 2100 proves to be non intrusive in that it did not require any changes to the game core logic.
- Although the implementation requires the addition of a number of control structures to the server, few changes were required on the client.
- VFC parameter definition can easily be externalized to a configuration file, enabling fast tunning and optimization of the VFC algorithm.

Chapter 5

Evaluation

In this chapter we present the experimental results of our evaluation of VFC-RTS. We start by presenting the conditions in which the evaluation took place (Section 5.1), followed by a quantitative evaluation (Section 5.2), consisting on a study on the use of both computational and network resources. Finally, in Section 5.3, we introduce a qualitative evaluation, in which we investigate how the application of the VFC-RTS framework affects a player's perception of the game.

5.1 Test Environment

In order to achieve a rich evaluation of our system, capturing a large variety of real game scenarios, our experiments were conducted varying different factors such as the number of players, the size of the game scenario and the number of game entities controlled by the players.

All experiments were performed by human players, from novice gamers with little background on RTS games, to experienced players, familiarized with the reality of multiplayer gaming. This factor contributed to a more realistic gaming environment since it was possible to assemble a test group with different game strategies and expectations. Although novice gamers choose for a more cautious approach, focusing primarily on reenforcing their base, slowly discovering the game scenario as their troops numbers increase, experienced gamers opt for a more aggressive strategy, sending their troops throughout the map as they become available, in order to quickly discover the opponents headquarters and attack them by surprise.

By testing VFC-RTS with players that employ different game strategies, we can make a better assessment of the system, since different strategies may result on different areas-of-interest. Thus, an occasional player's AoI would consist primarily on his base command area and surroundings, while the one of an experienced player would constantly be shifting.

The tests were performed on two to seven machines, depending on the number of players on the session, from a laptop computer with Intel Pentium M 1.8 GHz and 512 Mb of main memory to a Intel Core 2 Duo processor and 4 Gb RAM, all running Windows 7 32-bit OS. The server used for the tests was also a 2.6 GHz Intel Core 2 Duo with 4 Gb RAM running on Linux Ubuntu distribution. The tests were performed on both fixed and wireless networks.

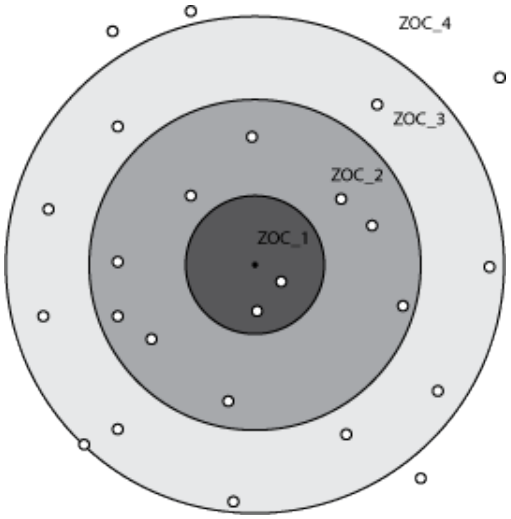


Figure 5.1: Zones of Consistency employed during testing

	Zones of Consistency			
	ZOC_1	ZOC_2	ZOC_3	ZOC_4
Time (s)	1	4	8	40
Sequence	1	3	5	10
Value (px)	1000	2000	4000	10000

Table 5.1: Values for the Zones of Consistency employed during testing

In Figure 5.1 we present a 2-dimensional view of the zone of consistency distribution employed during this evaluation, followed by the specification for each zone's consistency vector (Table 5.1). Analyzing the presented zones of consistency it is important to notice that ZOC_1, the zone with the most strict consistency requirements, entirely circumscribes the player's area-of-interest, assuring at all times the player is visually presented with the most accurate information about the game.

Also worth pointing out from the consistency zones analysis is ZOC_4. Consisting on the map area not inscribed on the other zones of consistency, ZOC_4 represents the minimum consistency degree a player must be presented with during the duration of the game. In spite of not representing a player's immediate area-of-interest, ZOC_4 is an important feature to maintain a player unaware of game state inconsistencies during a fast camera motion. The values for each parameter in the consistency vectors were determined during development stage.

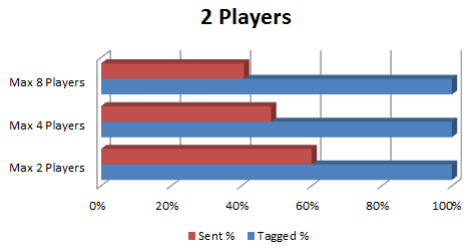
5.2 Quantitative Evaluation

In this section we show the results of a study conducted to the system with the objective of determining the savings in the use of network resources as well as the impact on computational performance. In order to evaluate our solution and determine whether it improves the scalability of the game we made use of metrics presented on *GSM: Game Scalability Model*[20, 21] such as monitoring message arrival and dispatch, and CPU consumption, however, the model is not applied in its entirety, as we selected the most relevant aspects. None of the following tests were repeated, so there was no selection of the most convenient results.

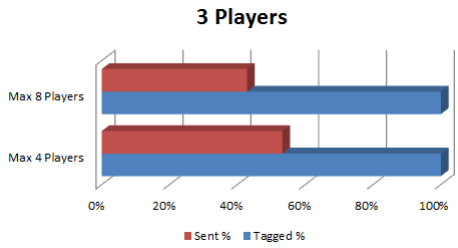
5.2.1 Number of Messages

We start our analysis by studying how applying the VFC algorithm affected server-client update propagation. In Figure 5.2, we present the ratio between the number of server update messages before and after applying VFC for two, three, and four players game sessions, varying for each case the size of the scenario (small, medium and large maps with a maximum capacity for two, four and eight players, respectively). The results are summarized on Figure 5.3.

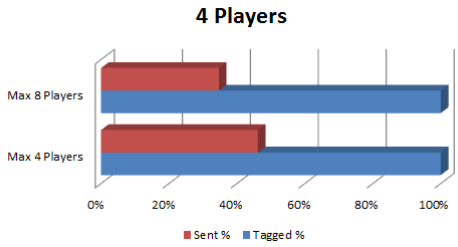
From a first analysis to the graphics, we are able to have overall reductions in the number of exchanged messages between 50% and 60%, considering an already interesting number of users. Such gains are not accomplished on a two players game in the small map (for only two players) since both bases are located too close to each other, resulting on an highly probable overlap between the player's areas-of-consistency. Figure 5.2 also illustrates how the size of the scenario has a direct impact on the update exchange rate. This factor can be explained by the increasing gap between player's headquarters as the scenario gets wider. Hence, the bigger the scenario, the more extensive is the distance between the player's AoI.



(a) Two players game



(b) Three players game



(c) Four players game

Figure 5.2: Comparing message dispatch ratio before and after applying VFC

Also worth of mentioning is the slight decline on the message exchange rate as the number of players increases. This results on the fact that, although the number of sent updates increases with the number of players, there is not a direct proportionality between these variables, as it would present previously to the implementation of our solution, since each client's game state update would have to be dispatched to every other client by the server. We believe that, as the number of players increases, the message exchange rate would gradually cease declining and stabilize.

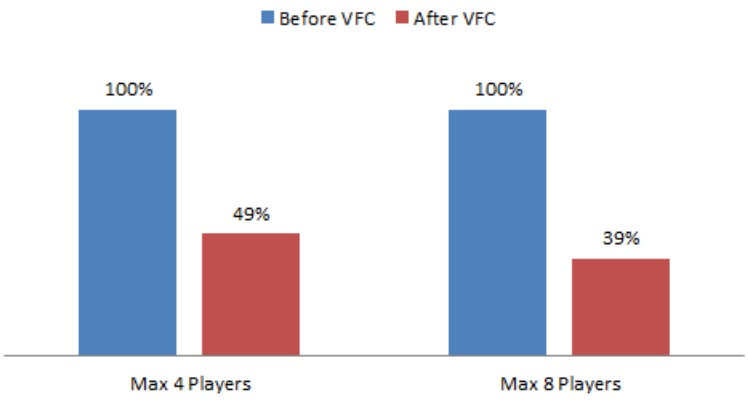
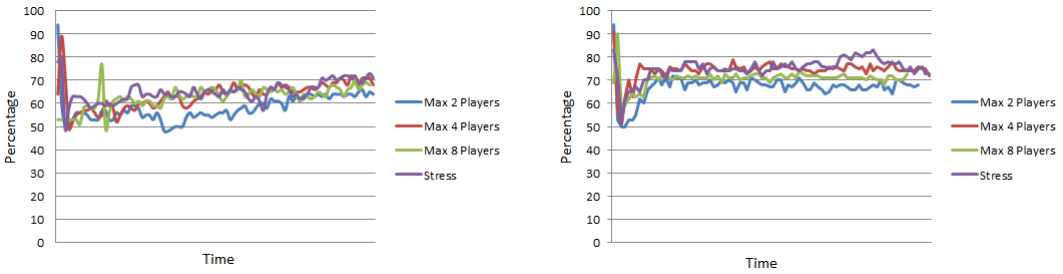


Figure 5.3: Overall message dispatch ratio before and after applying VFC

5.2.2 CPU Performance

In this section, we present the study the effects of our solution on CPU consumption. Since the majority of computational workload takes place on the server, we focus our study on this component.

In spite of being realized under the same variables as the previous test, to further examine the impact of our solution, we introduce a stress case. The stress case consists on a game session taking place on larger map where the players were enabled to make use of a game cheat that multiplies ten times the number of units controlled by him. With this test, we intend to make an assessment on how the system reacts considering a large number of game entities.



(a) Before VFC

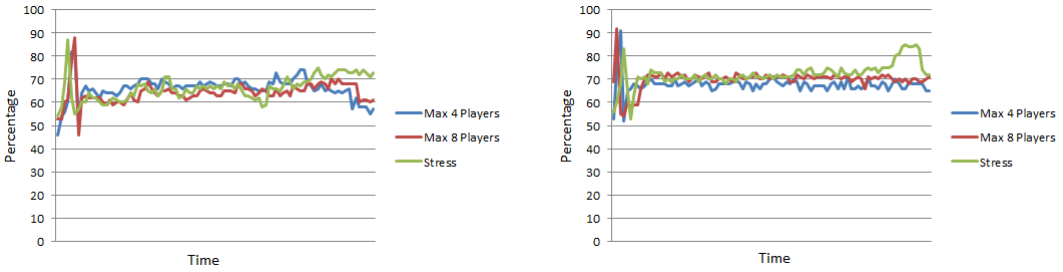
(b) After VFC

Figure 5.4: CPU consumption on a 2 players game

In the results, we can see an increase on the server CPU consumption in between 10 and 20 percent. This can be seen as a consequence of the additional workload required for enforcing the VFC algorithm. Despite this fact, such an increase on the server's computational requirements has little expression since CPU usage rarely surpasses 80%.

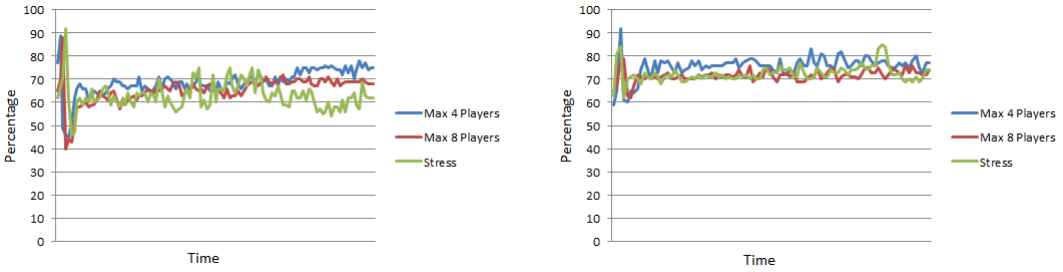
Additionally, there are two aspects worth mentioning from our graphical analysis. Although our solution presents additional CPU usage compared to the version without VFC, the consumption rate is consistent during the entire game session, in some cases independently of the map size (Figures 5.5(b) and 5.6(b)). Furthermore, it is important to notice the CPU consumption rate consistency even during the stress case.

It is also worth pointing out the CPU usage peak on Figure 5.5(b). This moment was documented



(a) Before VFC (b) After VFC

Figure 5.5: CPU consumption on a 3 players game

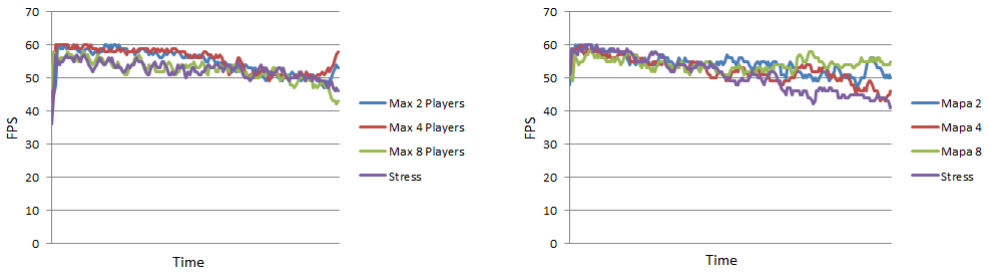


(a) Before VFC (b) After VFC

Figure 5.6: CPU consumption on a 4 players game

during the test as the moment when the three players were requested to battle each other on the center of the map, therefore further stressing the system. The result was a rise on CPU consumption from 70 to 75 percent and up to 85 percent. This situation proved that, in spite of momentarily requiring a critical use of computational resources, the system is able to handle a great number of entities.

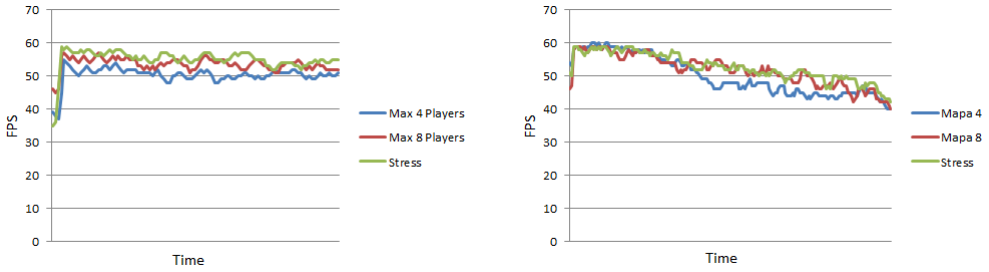
5.2.3 Frame Rate



(a) Before VFC (b) After VFC

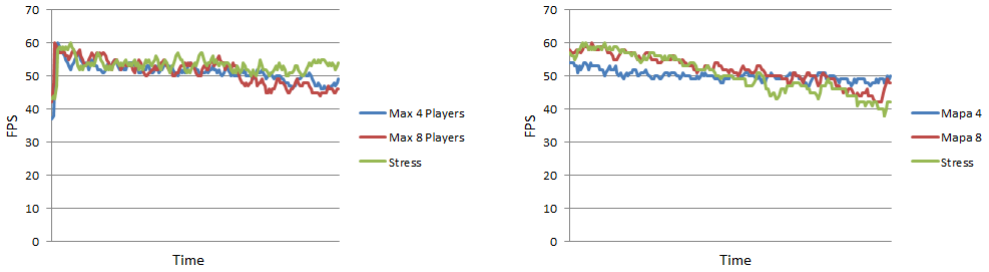
Figure 5.7: Frame rate on a game with 2 players

We proceed the quantitative analysis of our solution by monitoring the frame rate. Although this factor may sustain a direct influence on the player's game perspective, therefore also belonging in the



(a) Before VFC (b) After VFC

Figure 5.8: Frame rate on a game with 3 players



(a) Before VFC (b) After VFC

Figure 5.9: Frame rate on a game with 4 players

qualitative evaluation, in this section we will solely make an analysis on how the frame frequency was affected in the system. Figures 5.7, 5.8 and 5.9 illustrate the frame rate on a client during a two, three and four players session, respectively.

By looking at the charts it is easily perceptible that, although the frame rate is not uniform for the duration of the game neither before nor after applying VFC, the latter shows a slight downward trend, reaching a minimal value of 37 FPS (Figure 5.9(b)), while the former presents a minimal of 44 frames per second (Figure 5.9(a)). We believe these may come as a consequence of changes made to the update message structure in order to enforce the VFC algorithm. While, in the version before VFC, a message would simply propagate the object's latest orders, which would produce the same game state on all clients, since the system supports total consistency, the new update message structure also propagates the object's position when the order was given, so that the object may synchronize it's position with its primary replica and than proceed with the order, thus requiring additional rendering.

Although a minimal value of 37 frames per second was reached during a single stress situation, when analyzing remaining tests, we can see that the most common minimal value its about 40 frames per second, very close to the minimal frame rate before applying VFC.

Conclusions

Once concluded the quantitative evaluation to VFC-RTS we can infer that:

- Compared to the total consistency solution, VFC-RTS presents an overall reduction on the message exchange rate in the order of 50% to 60%, independent of the number of players.
- Although the server's CPU consumption shows an increase in the order of 10% to 20% when comparing to the total consistency solution, this increase shows little significance since the overall CPU usage rarely surpasses 80%.
- The frame rate on VFC-RTS shows a slight decay during game play, reaching an average minimal rate of 40 frames per second. However, since the minimal frame rate on the total consistency solution is about 44 frames per second, this difference on frame rate shows little significance.
- Ultimately, despite a 10% difference, we consider that VFC-related code could be optimized and achieve similar performance.

5.3 Qualitative Evaluation

In this work we improved a total-consistent multiplayer RTS game, enforced by the broadcast of all update messages, providing it with the benefits of the VFC model. The main objective of this qualitative study is to assess if the introduction to the VFC model had a significant impact on user experience.

To evaluate the quality impact of our solution, we have developed a questionnaire (consult Attachments at the bottom of this document) to which every player was subjected by the end of a Warzone 2100 game session. On the questionnaire we introduce the concept of *sudden object movement*. Consisting on an object's translation between two points without describing the path in between, sudden object movements are the visual manifestation of one of two situations: *lag* over the client-server communication channel or a game state correction due to momentarily unfulfillment of the consistency requirements over the player's AoI.

The questionnaire serves three purposes:

- Determining if the player experienced ***lag*** during the game, how often it would happen, the most frequent and maximum duration for the occurrences. Since *lag* occurrences can result on sudden object movements, it becomes an important factor to take into consideration during the results analysis.
- Determining if there were **sudden object movements** during gameplay, the most frequent distance travelled by the object during these events as well as the maximum distance.
- Determining if sudden object movements caused an **impact** on the player's game strategy, the frequency to which he had to adapt his strategy as a result of these events and whether or not he believes this factor influenced the outcome of the game.

On a first contact with the questionnaire, we have realized players had difficulty to understand the answer scale employed on the questions regarding the impact of sudden object movements on both game strategy and the final outcome. Due to this fact, we have presented the players with Tables 5.2 and 5.3, describing the evaluation metrics used for each question respectively.

The following tests were conducted on two, three, four and seven players game sessions. The two, three and four players session were performed over a fixed network while the seven players games over a wireless network.

Total	No strategy was fulfilled due to sudden object movements.
Severe	Sudden object movements had greater impact on gameplay than strategy itself.
Average	Although sudden object movements were noticeable during gameplay, game strategy could be appropriately adjusted.
Little	Sudden object movements have caused changes in strategy, however, they had little impact on gameplay.
None	Sudden object movements had no impact on game strategy.

Table 5.2: Evaluation metrics for question eleven of the Warzone 2100 Playability Questionnaire

Total	The outcome of the game was severely tampered by sudden object movements, giving no space for strategy .
Severe	Sudden object movements had greater influence on the outcome of the game than the game strategy itself.
Average	There is a 50% chance the outcome of the game was influenced by a number of sudden object movements.
Little	Although sudden object movements may have caused changes in strategy, they had little influence on the outcome of the game.
None	The outcome of the game would have been the same if there were no sudden object movements.

Table 5.3: Evaluation metrics for question twelve of the Warzone 2100 Playability Questionnaire

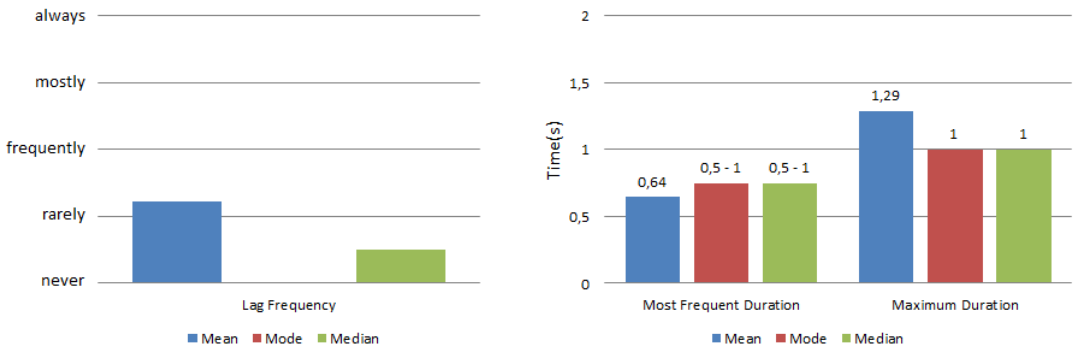
Lag

At this time we present the questionnaire results regarding *lag* experience. Despite not being a factor that determines the success of our solution by itself, these values provide us with a better understanding about the conditions in which the game sessions were conducted, thus enabling a more accurate evaluation of the work. During a preliminary result analysis we verified *lag* was only experienced during the seven players sessions, thus we will focus solely on these.

Figure 5.10 presents the mean, mode and median values for the results regarding the player's perception of *lag* during the game. Although the mean value for the frequency of *lag* occurrences indicates that these events were noticeable, they have presented an overall low impact (median), and most players did not experience it (based on the mode).

Additionally, in Figure 5.10(b), we can see that, among the players that actually experienced *lag*, who already were just a fraction, the most frequent duration was between 0,5 and 1 second, as the maximum duration values also stands close to 1 second.

As the seven players sessions were the only ones conducted under a wireless network, we believe this was the determining reason for the network latency verified in these tests.



(a) (b)

Figure 5.10: *Lag* experience on a 7 Players/WLAN session

Sudden Object Movements

We continue the qualitative evaluation of VFC-RTS by studying the players awareness of sudden object movements in the course of the game, starting by determining the frequency to which these events toke place (Figures 5.11 and 5.12).

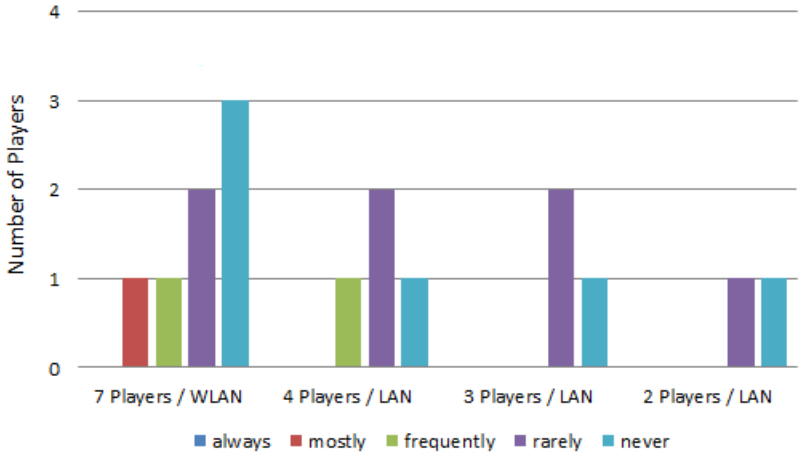


Figure 5.11: Sudden object movements frequency

By examining the graphics we can verify a slight increase on the frequency of sudden object movements as the number of players increases, however, the values for mean, median and mode do not reflect this fact, as these values fall between "never" and "rarely", regardless of the number of players.

Moreover, we can detect a clear distinction between the results regarding the LAN sessions and the WLAN sessions. Although LAN game sessions present a consistent sudden object movements frequency increase pattern as the number of players increases, the WLAN sessions results present an irregular frequency growth. Since these sessions were the only ones where latency was verified, we believe this fact may have influenced the results.

Additionally, it is important to notice that, although sudden object movements were noticed by most players, just a small number of them classify these occurrences as frequent.

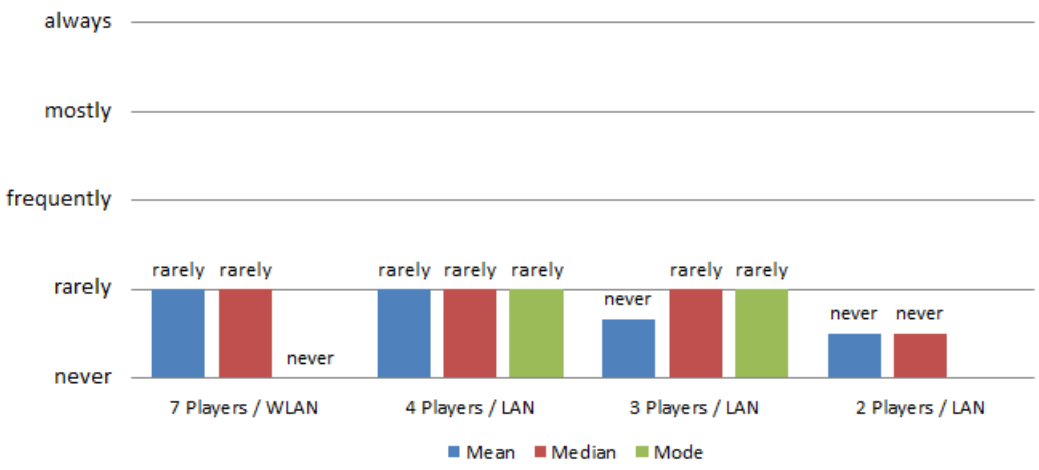
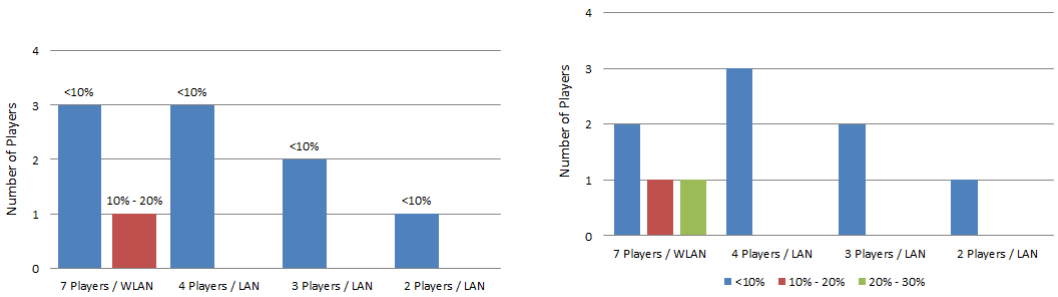


Figure 5.12: Sudden object movements frequency statistics



(a) Most frequent

(b) Maximum

Figure 5.13: Distance travelled during a sudden object movement

On Figure 5.13 we present the results for most frequent and maximum distance travelled by objects that presented sudden movements. By examining the charts we can tell they present very similar results, as they both show the majority of players noticing sudden object movements say the distance travelled equals less than 10% of the visible game scenario, which represents an overall short distance. Once again we verify that the sessions performed over a wireless network present a different result set comparing to the sessions performed over a physical network, presenting cases of most frequent travelled distance in the order of 10% to 20% and even a maximum travelled distance case of 20% to 30% of the visible scenario.

Impact on the game

To complete the qualitative study of the solution, in this section we will analyze how the occurrence of sudden object movements has influenced the players strategies. Figures 5.14 and 5.15 present the results for how often the players had to make strategy adjustments due to sudden object movements and how much of an impact did they produce on overall game strategy.

Although coherent, comparing to the results on the previous section, the differences between the result sets on the seven players games and the ones of the remaining games are aggravated when evaluating the impact of sudden object movements on game strategy. We believe this aspect is deeply linked to the RTS gaming background of the players involved in these tests. While a player experienced on RTS

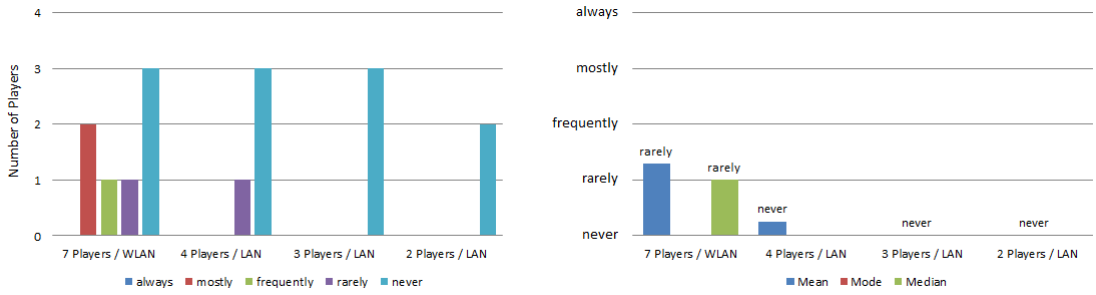


Figure 5.14: Game strategy adjustment frequency

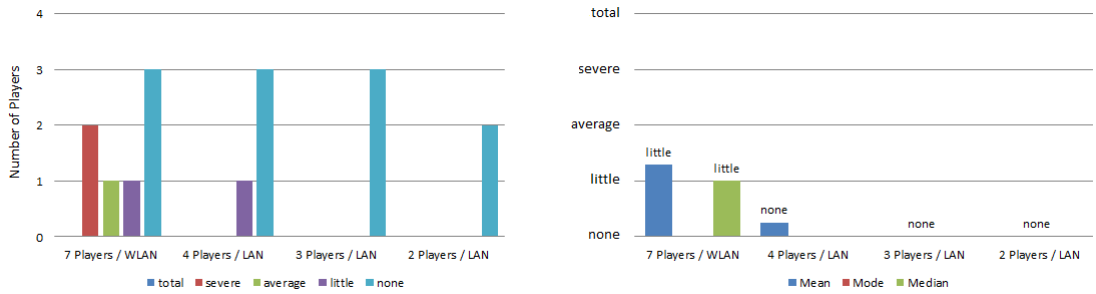


Figure 5.15: Impact on game strategy

games have preconceived strategies on how to approach a game and achieve victory, a novice player tend to form a strategy during the course of the game. Although an experienced player approach may prove to hold better chances on winning, it is not as flexible or adaptable as the one of a novice player, hence, sudden object movements may require more frequent strategy adjustments, causing a greater impact on the game strategy. The same way, a player with little RTS experience may not consider as much strategy adjustments as he only develops short term strategies.

In accordance with our previous studies, the two, three and four players sessions, which present a low sudden object movement frequency, are also the ones which present the lowest frequency of game strategy adjustments, thus producing no impact on game strategy.

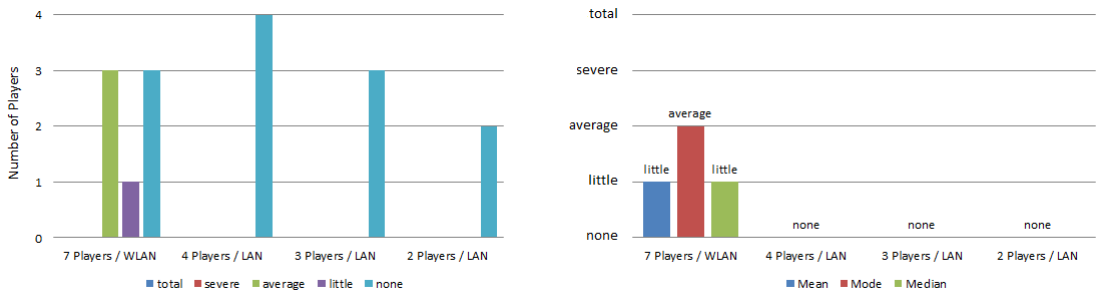


Figure 5.16: Impact on the outcome of the game

Moreover, analyzing the mean, mode and median values for game strategy adjustment frequency, we verify that only the seven players sessions present significant statistical values, with "rarely" as mean and median, which in turn result on "little" impact on game strategy (Figure 5.15).

Finally, on Figure 5.16, we present the chart with the results on how much have sudden object movements influenced the outcome of the game.

In coherence with the results described on Figure 5.15, we can notice that, on the two, three and four players sessions, where sudden object movements presented no significant impact on game strategy, all players agree that the outcome of the game was in any way influenced by these events. However, on the seven players sessions, the impact caused on game strategy reflects itself on the game's outcome, which may once more have been influenced by the level of experience on RTS games the players present.

Nevertheless, it is interesting to verify that the magnitude of the impact caused by sudden object movements to the outcome of the game is smaller than the magnitude of the impact caused to game strategy, which suggests that strategical decision is still the decisive factor on the outcome of the game. Taking a closer look over the mean, mode and median values on Figure 5.15 we verify that although the mode values are both "little" and "average", the mean and median values are both "little", which confirms the premise that strategy is the most determinant aspect on the outcome of the game

Conclusions

Upon the completion of the qualitative evaluation of VFC-RTS we can conclude that:

- Although the latency verified during the seven players sessions, conducted over a wireless network, may have tampered the result of this study by causing additional sudden object movements, these events caused little impact on the player's strategy, hence on the outcome of the game.
- The game sessions conducted over a fixed network also presented sudden object movements, however, the frequency of these occurrences was lower than on the game sessions over a wireless network, and presented no impact to the player's strategy or the game's outcome.
- VFC-RTS holds little to none overall impact to the player's game experience, however, this value is subject to the latency of the system.

Chapter 6

Conclusion

In this document, we have presented a continuous consistency model which resulted from the adaptation of the VFC model to the RTS multiplayer environment. Besides providing a continuum between strong and weak consistency, the VFC model makes use of the notion of *locality-awareness* to define multiple zones of consistency with different consistency degrees. By doing so, the VFC model enables selective updating in accordance to a user's area-of-interest.

To start this work, we provided an overview upon the state of the art on three fields: consistency maintenance in distributed systems, interest management techniques and RTS games.

Next, we presented the architecture for a system powered with the adaptation of the VFC model, describing the adjustments that, in our opinion, were required in order to best adjust the model to the real-time strategy paradigm. In this adaptation, we defined the portion of the scenario captured by the player's camera as being his primary area-of-interest, therefore, the zone of consistency with the most strict consistency requirements, describing a *pivot point* at its center. Moreover, to guarantee that the player is presented at all times with the most accurate information about events regarding his troops, even if located outside the area covered by the camera, we introduced the concept of *unit*, a group of entities presenting a pivot point at its geometric center. An unit allows following entities outside the camera span without compromising the consistency of the system.

The defined architecture was implemented on top of *Warzone 2100*, an open-source RTS game with total consistency requirements. After the implementation, using a number of criteria, we analyzed the success of this work. In addition to showing the correctness of the adaptation and the little impact caused to the player's game experience, the results showed the great potential of the VFC model. In comparison to the network resources usage on a total consistency system, we were able to achieve reduction by the order of fifty percent. This way, a VFC powered RTS game can maintain strict, locality based, consistency levels without compromising the player's perception of the game, an advantageous feature to games with a large number of simultaneous gamers such as MMOGs.

6.1 Future Work

In this work we have established the grounds for a wide variety of future works, including extensions and improvements to the current design and implementation. In this subsection we enumerate some of the

lines of investigation we intend to pursue:

- Extend the currently centralized architecture of VFC-RTS to support a distributed server infrastructure. Since the current solution relies on a single server node to accept all client connections and enforce the VFC algorithm through the entire extension on the scenario, a possible improvement to the solution would be dividing the scenario into smaller areas and assigning each one to a different server. This way, we would achieve greater load balancing by dividing the VFC reasoning workload through the servers present in the system. Furthermore, this would also increase the scalability of the system as every server would be able to accept client connections.
- Enhance the consistency zones monitoring system. Currently, the VFC-RTS solution solely monitors changes to the position of the player's camera, however, modern 3-dimensional RTS games enable additional camera operations such as zoom and rotation, which we intend to take into consideration on future work.
- Explore the application of the VFC-RTS to a world wide global positioning system. As total consistency is impossible to accomplish on a global scale, a location aware solution such as VFC-RTS would prove to be a viable candidate.
- Expand the use of the VFC-RTS architecture to other online multiplayer game types that present a top-down camera perspective such as race car driving.

Bibliography

[1] D. Ahmed, S. Shirmohammadi, and J. de Oliveira. A hybrid p2p communications architecture for zonal mmogs. *Multimedia Tools and Applications*, 45:313–345, 2009. 10.1007/s11042-009-0311-y.

[2] D. T. Ahmed and S. Shirmohammadi. A dynamic area of interest management and collaboration model for p2p mmogs. In *Proceedings of the 2008 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*, DS-RT '08, pages 27–34, Washington, DC, USA, 2008. IEEE Computer Society.

[3] J. a. Barreto, J. a. Garcia, L. Veiga, and P. Ferreira. Data-aware connectivity in mobile replicated systems. In *Proceedings of the Eighth ACM International Workshop on Data Engineering for Wireless and Mobile Access*, MobiDE '09, pages 9–16, New York, NY, USA, 2009. ACM.

[4] C. E. Bezerra, F. R. Cecin, and C. F. R. Geyer. A3: A novel interest management algorithm for distributed simulations of mmogs. In *Proceedings of the 2008 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*, DS-RT '08, pages 35–42, Washington, DC, USA, 2008. IEEE Computer Society.

[5] 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, New York, NY, USA, 2006. ACM.

[6] J. Carvalho, J. Oliveira, and P. Carvalho. Jogos on-line: Estudo sobre qoe e qos. In *CRC 2010 - 10a Conferência sobre Redes de Computadores*, pages 83–88, Universidade do Minho, Braga, Portugal, 2010. CRC 2010.

[7] M. Claypool. The effect of latency on user performance in real-time strategy games. *Computer Networks*, 49(1):52 – 70, 2005. Networking Issue in Entertainment Computing.

[8] J. F. F. d. Costa. Vector-field consistency for cooperative work. In *MSc Thesis 2010*, Instituto Superior Técnico, Lisboa, Portugal, 2010.

[9] J. Dyck, C. Gutwin, T. C. N. Graham, and D. Pinelle. Beyond the lan: techniques from network games for improving groupware performance. In *Proceedings of the 2007 international ACM conference on Supporting group work*, GROUP '07, pages 291–300, New York, NY, USA, 2007. ACM.

[10] C. A. Ellis and S. J. Gibbs. Concurrency control in groupware systems. *SIGMOD Rec.*, 18:399–407, June 1989.

[11] T. A. Funkhouser. Ring: a client-server system for multi-user virtual environments. In *Proceedings of the 1995 symposium on Interactive 3D graphics*, I3D '95, pages 85–ff., New York, NY, USA, 1995. ACM.

[12] T. Hampel, T. Bopp, and R. Hinn. A peer-to-peer architecture for massive multiplayer online games. In *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*, NetGames '06, New York, NY, USA, 2006. ACM.

[13] B. Knutsson, M. M. Games, H. Lu, W. Xu, and B. Hopkins. Peer-to-peer support for massively multiplayer games, 2004.

[14] N. Krishnakumar and A. J. Bernstein. Bounded ignorance: a technique for increasing concurrency in a replicated system. *ACM Trans. Database Syst.*, 19:586–625, December 1994.

[15] D. d. M. T. Lage. Vector-field consistency for .net multiplayer games. In *MSc Thesis 2010*, Instituto Superior Técnico, Lisboa, Portugal, 2010.

[16] L. Lamport. Time clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21:558–565, July 1978.

[17] E. Lety, L. Gautier, and C. Diot. Mimaze, a 3d multi-player game on the internet. In *Proceedings of the 4th International Conference on Virtual System and MultiMedia*, 1998.

[18] F. W. Li, L. W. Li, and R. W. Lau. Supporting continuous consistency in multiplayer online games. In *Proceedings of the 12th annual ACM international conference on Multimedia*, MULTIMEDIA '04, pages 388–391, New York, NY, USA, 2004. ACM.

[19] B. F. Loureiro. Vfc-game. In *MSc Thesis 2010*, Instituto Superior Técnico, Lisboa, Portugal, 2010.

[20] J. Müller and S. GORLATCH. Gsm: a game scalability model for multiplayer real-time games. *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, 00:2044–2055, 2005.

[21] J. Müller and S. GORLATCH. Rokkatan: scaling an rts game design to the massively multiplayer realm. *Comput. Entertain.*, 4, July 2006.

[22] J. Munson and P. Dewan. A concurrency control framework for collaborative systems. In *Proceedings of the 1996 ACM conference on Computer supported cooperative work*, CSCW '96, pages 278–287, New York, NY, USA, 1996. ACM.

[23] A. Negrão. Vfc large-scale: Consistency of replicated data in large scale networks. In *MSc Thesis 2009*, Instituto Superior Técnico, Lisboa, Portugal, 2009.

[24] D. A. Nichols, P. Curtis, M. Dixon, and J. Lamping. High-latency, low-bandwidth windowing in the jupiter collaboration system. In *Proceedings of the 8th annual ACM symposium on User interface and software technology*, UIST '95, pages 111–120, New York, NY, USA, 1995. ACM.

[25] L. Pantel and L. C. Wolf. On the impact of delay on real-time multiplayer games. In *Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*, NOSSDAV '02, pages 23–29, New York, NY, USA, 2002. ACM.

[26] D. S. Parker, G. J. Popek, G. Rudisin, A. Stoughton, B. J. Walker, E. Walton, J. M. Chow, D. Edwards, S. Kiser, and C. Kline. Detection of mutual inconsistency in distributed systems. *IEEE Trans. Softw. Eng.*, 9:240–247, May 1983.

[27] J. D. Pellegrino and C. Dovrolis. Bandwidth requirement and state consistency in three multiplayer game architectures. In *Proceedings of the 2nd workshop on Network and system support for games*, NetGames '03, pages 52–59, New York, NY, USA, 2003. ACM.

[28] Y. Saito and M. Shapiro. Optimistic replication. *ACM Comput. Surv.*, 37:42–81, March 2005.

[29] N. Santos, L. Veiga, and P. Ferreira. Vector-field consistency for ad-hoc gaming. In R. Cerqueira and R. Campbell, editors, *Middleware 2007*, volume 4834 of *Lecture Notes in Computer Science*, pages 80–100. Springer Berlin / Heidelberg, 2007.

[30] M. Shapiro and N. Preguia. Designing a commutative replicated data type. Technical report, Computer Science Dept: University of Copenhagen, 2007.

[31] S. Singhal and M. Zyda. *Networked virtual environments: design and implementation.* ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1999.

[32] J. Smed, T. Kaukoranta, and H. Hakonen. A review on networking and multiplayer computer games. In *In multiplayer computer games, proc. int. conf. on Application and development of computer games in the 21st century*, pages 1–5, 2002.

[33] C. Sun and C. Ellis. Operational transformation in real-time group editors: issues, algorithms, and achievements. In *Proceedings of the 1998 ACM conference on Computer supported cooperative work*, CSCW '98, pages 59–68, New York, NY, USA, 1998. ACM.

[34] C. Sun, X. Jia, Y. Zhang, and Y. Yang. A generic operation transformation scheme for consistency maintenance in real-time cooperative editing systems. In *In Procs. of Intl. ACM SIGGROUP Conf. on Supporting Group Work*, pages 425–434. Press, 1997.

[35] C. Sun, Y. Y., Z. Y., and C. D. A consistency model and supporting schemes for real-time cooperative editing systems. In *Proceedings of the 19th Australian Computer Science Conference*, pages 582–591, 1996.

[36] D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser. Managing update conflicts in bayou, a weakly connected replicated storage system. In *Proceedings of the fifteenth ACM symposium on Operating systems principles*, SOSP '95, pages 172–182, New York, NY, USA, 1995. ACM.

[37] A. Tumbde. A voronoi partitioning approach to support massively multiplayer online games. Technical report, The University of Wisconsin, 2004.

[38] L. Veiga, A. Negro, N. Santos, and P. Ferreira. Unifying divergence bounding and locality awareness inreplicated systems with vector-field consistency. *Journal of Internet Services and Applications*, 1:95–115, 2010. 10.1007/s13174-010-0011-x.

[39] J. Vogel and M. Mauve. Consistency control for distributed interactive media. In *Proceedings of the ninth ACM international conference on Multimedia*, MULTIMEDIA '01, pages 221–230, New York, NY, USA, 2001. ACM.

[40] A. P. Yu and S. T. Vuong. Mopar: a mobile peer-to-peer overlay architecture for interest management of massively multiplayer online games. In *Proceedings of the international workshop on Network and operating systems support for digital audio and video*, NOSSDAV '05, pages 99–104, New York, NY, USA, 2005. ACM.

[41] H. Yu and A. Vahdat. Design and evaluation of a continuous consistency model for replicated services. *ACM Trans. Comput. Syst.*, 20:239–282, August 2002.

Warzone 2100 Playability Questionnaire

This survey intends to capture the quality of experience of a player during a session of Warzone 2100. If you have any questions while filling out the survey please submit them to a monitor.

Thank you.

About the game session

1. **What Warzone 2100 version did you player?**
   - ☐ Master version
   - ☐ VFC version
2. **What was your player name?** _____
3. **How long did the game session last?** _____ minutes

During the game

Did you experience lag[1]?

4. **How often did you experienced lag?**
   - ☐ always
   - ☐ mostly
   - ☐ frequently
   - ☐ rarely
   - ☐ never

5. **What was the most frequent lag duration?**
   - ☐ 0 seconds
   - ☐ 0,1 - 0,5 seconds
   - ☐ 0,5 - 1 second
   - ☐ >1 second

6. **What was the maximum lag duration?** _____ seconds

Did you notice any sudden object movements?

(if you did not notice sudden object movements jump to question 10)

7. **How often did you notice sudden object movements?**
   - ☐ always
   - ☐ mostly
   - ☐ frequently
   - ☐ rarely
   - ☐ never

8. **What was the most frequent distance traveled during a sudden movement?**

   (considering the visible map area)
   - ☐ <10%

[1] Lag: Delay (or latency) between an action by a player and the reaction of the game

- ☐ 10% - 20%
- ☐ 20% - 30%
- ☐ 30% - 40%
- ☐ 30% - 50%
- ☐ >50%

9. What was the maximum distance traveled during a sudden movement?

(considering the visible map area)

- ☐ <10%
- ☐ 10% - 20%
- ☐ 20% - 30%
- ☐ 30% - 40%
- ☐ 30% - 50%
- ☐ >50%

Do you believe sudden object movements influenced your game strategy?

10. How often did it influence your game strategy?

- ☐ always
- ☐ mostly
- ☐ frequently
- ☐ rarely
- ☐ never

11. Had it an impact in your game strategy?

- ☐ total
- ☐ severe
- ☐ average
- ☐ little
- ☐ none

12. Do you believe it influenced the outcome of the game?

- ☐ total
- ☐ severe
- ☐ average
- ☐ little
- ☐ none