From 101 To nnn: A Review and A Classification of Computer Game Architectures

Meng Zhu

Alf Inge Wang

Hong Guo

Department of Computer and Information Science,
Norwegian University of Science and Technology, Sem Sælands Vei 7-9, NO-7491, Trondheim, Norway

Abstract The Game Worlds Graph (GWG) framework is a taxonomy for analyzing and classifying computer game architectures. This article presents a systematic review of game architectures using the GWG framework. The review validates the usefulness of the GWG framework through classifying game architectures described in the literature into distinct categories according to the framework. The major contribution of the paper is a state-of-the-art presentation of 40 different game architectures, which covers architectures for all kinds of games from single player games to Massively Multiplayer Online Games (MMOGs). Previous reviews of game architectures have focused on a narrower selection of games such as only networked games, MMOGs or similar. Further, none of the previous reviews has used a systematic framework for analyzing the characteristics of game architectures. By using the framework, we can identify similarities and differences of the 40 game architectures in a systematic way. Finally, the paper outlines the evolution of the game architectures from the perspective of the GWG framework.

Keywords*—GWG (Game Worlds Graph), Software Architecture, Computer Game*

1. Introduction

The software architecture of a computer game influences both the quality attributes and the functionality, and has therefore played an important role in game development. Today, game designers and programmers have the opportunities to create new social and innovative games using technologies like the Internet and hardware and services provided by mobile computing, which drives the evolution of game architectures. Game architectures are getting larger, more complex and more sophisticated, evolving from stand-alone to Massively Multiplayer Online Game (MMOG) architectures. The expansion and higher complexity of architectures introduce new challenges. Research within game architectures addresses issues like state consistency, anti-cheating, load balancing, and so forth. Such issues require research that examines the detailed design of various game architecture approaches, thus make it important to have a descriptive

taxonomy framework, which is capable of classifying different architectures in sufficient detail. We have created the Game Worlds Graph (GWG) framework for this purpose [48], which is a systematic framework designed not only for classifying and analyzing existing architectures, but also for exploring potentially new and unknown architectures. An application of the framework is presented in this article, which focuses on giving an overview and a classification of existing game architectures in literature using the GWG framework. The results presented in this paper have been achieved by conducting a systematic review of research articles, where game architectures described in these articles are analyzed, and then categorized according to the GWG framework. Each identified architectural category is discussed along with advantages and challenges of this category based on its GWG attributes. The paper also presents the major architectural characteristics of each reviewed architecture.

We have found that the existing related works are restricted to one specific domain of games with emphasis on network games or more specifically MMOGs. Systematic review as a research method has not been used before on reviews of game architectures. In [1], Jiang et al. have reviewed software architectures of network games. They classified architectures in three categories: Distributed Server Architecture, Hybrid Peer-to-Peer Architecture, and Structured Peer-to-Peer Architecture. The article discusses the characteristics of architectures with the focus on design tactics to provide scalability. The review has a narrower scope and a more specific focus than our work. [2] is another review article which focuses on techniques for Peer-to-Peer MMOGs. The authors mainly use six design issues related to Peer-to-Peer MMOGs as their framework with which design tactics are discussed. Since these tactics are elements of software architecture, the review is performed at a finer granularity. To illustrate the design tactics, six representative architectures are analyzed, which are all Peer-to-Peer architectures. Compared to the related work described above, our review is different in at least three aspects:

1) We have used the systematic review research method, which results in a better coverage of the literature.
2) We have used the GWG framework as the review framework, which categorizes game architectures according to Game World decomposition and connections. The framework provides an approach for characterizing game architectures found in the literature according to main characteristics, without getting into low-level technical details.
3) We have included game architectures for *all kinds* of computer games in our review, which is not restricted to one type of games or one type of architectures.

Based on the arguments above, our systematic review draws a more complete picture of game architectures covering innovations on architectures of *all kinds* of computer games, from stand-alone games to MMOGs, which has not been presented in any previous articles in the literature.

The remaining of the article is organized as follows: Section 2 introduces the Game Worlds Graph framework, Section 3 presents utilized research method and design, Section 4 presents the results, Section 5 discusses major findings in the research, and Section 6 concludes the article.

2. Brief Description of the GWG Framework

This section gives an overview of the GWG framework. A more detailed definition and description of the GWG framework can be found in [48].

2.1 Game Worlds Graph

The Game Worlds Graph (GWG) framework is based on two core concepts: *Game Worlds* and *World Connectors*. A *Game World* is a metaphor used to describe the game state of a computer game. The GWG framework identifies three kinds of Game Worlds in a computer game:

1) *Global World:* The Global World (GW) contains the shared global state (all player characters, non-playable characters (NPCs), weapons, etc.) of a game instance. The GW of a game reflects the definite, instant, and complete result of a game computation (the state of the game at a given moment when running the game).
2) *Local World:* The Local World (LW) can be regarded as the "cached" game state for one specific player. The LW contains only the game state that the game software considers "relevant" to a specific player. The game state reflected by the LW is a *dirty* state, which could be outdated or up-to-date, correct or incorrect, complete or incomplete, depending on the client computation and the synchronization protocols. The LW is usually used to keep a game sufficiently interactive, to save bandwidth, and to prevent cheating.
3) *Perceptible World:* The Perceptible World (PW) is the Game World experienced on the player's screen and/or other presentation devices. The PW can only contain a very limited subset of the GW or the LW due to rules of the game and performance concerns. Players perceive the game only through the PWs externalized by various presentation devices (screens, loudspeakers etc.).

The Game Worlds are not isolated and the content of a Game World is usually connected to the content of another Game World. For instance, a LW on a client needs to be consistent with the GW on the server. We use the term *World Connector* to capture the relationship between two connected Game Worlds. A World Connector can be either *native* or *remote*, depending on whether the two connected Game Worlds are simulated on the same node (machine) or distributed on two separated nodes respectively. The GWG framework defines two kinds of connectors connecting the three types of worlds (global, local and perceptible):

1) *Native Connector:* Denotes the computational synchronizing of two Game Worlds on the same computing node.
2) *Remote Connector:* Denotes the computational synchronizing of two Game Worlds on two different computing nodes.

The GWG framework uses graph theory to model and relate terms, where Game Worlds and World Connectors are considered as vertices and edges. The Game Worlds Graph (GWG) is formalized as follows:

*GWG = (**W**, **C**), where **W** is the set of Game Worlds simulated by a game instance, and **C** is the set of World Connectors connecting elements in W.*

2.2 Graph Styles and Flavors

Graph Styles and Flavors are used in the GWG framework to group game architectures that fits into the same category. If two GWGs have the same *multiplicities* for all three kinds of Game Worlds (global, local and perceptible), they have the same *combination* of Game Worlds multiplicities. The term *Graph Style* is used to capture the commonality of a group of GWGs, which share the same combination. By using 0, 1, and n to represent *zero* Game World, *one* Game World and *multiple (0 to n)* Game Worlds respectively, various configurations can be represented. Thus a Graph Style can be captured with the following expression: $G_nL_nP_n$, where G_n is the multiplicity of GWs (global), L_n is the multiplicity of LWs (local), and P_n is the multiplicity of PWs (perceptible). The valid set of Graph Styles that represents games that makes sense is the following: {*101, 10n, 111, 11n, 1n1, 1nn, n01, n0n, n11, n1n, nn1, nnn*}. These Graph Styles are valid because according to the common sense, a computer game should at least have a global state, and a perceptible output.

The types of connectors between two kinds of Game Worlds denote the game state distribution among computational nodes. If connectors between any GW and LW in a game are all remote connectors or are all native connectors, we say the GWs and the LWs are *remotely* connected or *natively* connected respectively. The term GWG *Flavor* describes the combination of Connector Type and Graph Style. The graph Flavor concept is an elaboration of the Graph Style concept that includes the Connector Type. GWG Flavors can also be captured with expressions formed by following these principles:

1) The Graph Style expression is used as the base to form the GWG Flavor expression.
2) When LWs are not presented in the GWG Flavor, the notation for LWs in the Graph Style expression is skipped. E.g., *1n* means one GW and multiple PWs (no LWs).
3) When two kinds of Game Worlds are natively connected, the Connector Type notation is skipped. E.g., *1n* represents one GW that is *natively* connected to multiple PWs.
4) Use - to represent remote Connector Type. E.g. *1-nn* means one GW is *remotely* connected to multiple LWs and the latter are *natively* connected to multiple PWs.
5) Use (-) to represent that the same kind of Game Worlds are remotely connected to themselves. E.g., *n(-)-nn* means multiple GWs are *remotely* connected to each other, and they are *remotely* connected to multiple LWs, which are *natively* connected to multiple PWs.

2.3 Use GWG as a Taxonomy Framework for Game Architectures

The GWG framework should be able to identify the Graph Style and the Flavor of every existing game architecture. So we can classify game architectures by filling game architectures with the same Graph Style into the same category. The classification is graph-style-based, i.e., each category features a specific Graph Style. Another classification based on the Flavor can also be carried out similarly. These two classifications form two layers of granularity of the GWG taxonomy framework. In this article, 40 game architectures described in research articles were first reviewed, then fitted into the GWG framework, and thus categorized. The resulting categorization

shows the feasibility of using the GWG framework as a taxonomy framework for classifying game architectures.

3. Research Method and Design

We adapted the systematic review method used in [3] in our research, which organizes the review through five distinct stages: *development of review protocol*, *identification of inclusion and exclusion criteria*, *search for relevant studies*, *critical appraisal*, *data extraction*, and *synthesis*.

Since our purpose was to review and classify existing architectures, quality evaluation of the reviewed architectures was irrelevant. This means that the *critical appraisal* stage could be skipped. Therefore our review was conducted according to these four stages:

1) The development of review protocol;
2) The identification of inclusion and exclusion criteria;
3) A search for relevant studies;
4) Solutions' extraction and classification.

These stages are described in detail in the remaining of this section.

3.1 Protocol Development

We developed a protocol by studying the protocol used in [3], and tailored it through a dedicated discussion among the authors. This protocol specifies the search strategy, inclusion and exclusion criteria, method of solutions extraction and classification.

3.2 Inclusion and Exclusion Criteria

The purpose of the research was to review and classify game architectures in literature. Articles that described one or more architectures were included. This included two types of articles:

1) Articles that claimed to propose one or more innovative architecture solutions.
2) Articles that described one or more architectures in sufficient detail, but the architectures had been proposed in an earlier published article.

Work of both researchers and practitioners were included. The review included articles from 1990, and up to and including 2009 (since we search the databases in 2010, articles published in 2010 and later were not included). Only work written in English was included. The architectures that were included in the review had to be architectural descriptions that covered all main parts of the software necessary for representing a computer game.

Articles were excluded if they did not provide sufficient details of the described architecture(s) to analyze them. Articles were excluded if they only presented a partial architecture of game software, e.g. rendering engine architecture, client architecture. Articles were excluded if they presented an architecture that included system(s) other than the gaming system itself, e.g. payment system, user management system, news system, etc. Articles were excluded, if they described an architecture that had previously been proposed in another article, while the latter had been included in the review (to avoid duplicated architectures).

3.3 Search for relevant studies

Due to time and resource constraints, only electronic databases were searched. The following four electronic databases were selected according to our knowledge of the field and the time budget:

- ACM Digital Library
- Compendex
- IEEE Xplore
- ScienceDirect –Elsevier

The following keywords were used:

(*Game* OR *Gaming*) AND *Architecture*

"Game" and "Gaming System" are two popular names for game software. "Architecture" can mean different concepts such as building and network architectures, but there is no other keyword that is more specific. The keywords were searched in titles, abstracts and indexes. Four databases were searched, resulting in a total of 2766 "hits". Then the authors went through these findings and excluded irrelevant articles by looking at the title, the publication source, and the abstract. The papers that had not been excluded were imported to the EndNote bibliography tool. Duplicated papers were excluded. A total of 154 articles were considered as potentially relevant after this step. All 154 candidates had to be examined carefully by the authors by studying the full texts of the articles, excluding articles according to the inclusion and exclusion criteria, and a total of 40 articles were selected for reviewing.

3.4 Architecture Solutions Extracting and Classification

For this stage a table with all 14 valid Graph Styles was created forming 14 architectural categories. Then the game architectures described in the 40 articles were analyzed and classified into these 14 categories in the table. Further, the architectures were analyzed according to the GWG Flavors, and placed into corresponding sub-categories. If a sub-category did not exist from previous analysis, a new sub-category for the GWG Flavor was created and the architecture was placed into this sub-category.

After this characterization process was finalized, a tabular view of the architectures was produced describing the reviewed articles categorized according to Graph Styles and Flavors. The next step was to further analyze and discuss the reviewed architectures in relation to their Graph Styles and GWG Flavors. This analysis resulted in a summary of the challenges and advantages connected to each category and sub-category according to our experience and to existing literature. This not only shows the reasoning power of the GWG framework, but also gives the big picture of game architecture research. The review results are reported in Section 4.

4. The Review Result

The 40 reviewed architectures fall into 5 Graph Styles, and 13 GWG Flavors. Table 1 shows a tabular view of how the architectures are distributed according to the Graph Style and Flavor, and these architectures are represented with the corresponding sources.

Table 1 Classification of Architectures According to the GWG framework

Graph Style	**GWG Flavor**	**Sources**
101	*1-1*	[6]
	11	[7]
10n	*1-n*	[8]
	1n	[9]
111 / *11n* / *1n1*	N/A	
1nn	*1-nn*	[10], [11], [12]
	1-n-n	[13]
	1-nn-	[14]
	1-n(-)n	[15], [16]
n01	N/A	
n0n	*n(-)n*	[17], [18], [19], [20], [21], [22], [23], [24], [25]
n11 / *n1n* / *nn1*	N/A	
nnn	*n-nn*	[26], [27], [28], [29], [30], [31], [32], [33], [34], [35]
	n(-)-nn	[36], [37], [38], [39], [40], [41]
	n-n(-)n	[42], [43], [44]
	n(-)-n(-)n	[45]

From Table 1 we can see that most of the architectures fall into *1nn*, *n0n* and *nnn* Graph Styles and *n(-)-nn*, *n(-)n*, and *n-nn* Flavors. Note that among the 40 architectures, 38 architectures fall into GWG Flavors with at least one *remote* Connector Type. This shows that the game architecture research mainly focuses on network game architectures.

To make the Flavor expressions easier to understand, we analyzed the major architectural characteristics of each GWG Flavor and proposed an alias system as shown in Table 2, where the terms used in the reviewed articles are also summarized. These terms are usually based on names for architectural patterns, e.g. Client-Server, Peer-to-Peer.

Table 2 Mapping the Terms Describing Game Architectures to the GWG Flavors

GWG Flavor	**Flavor Alias**	**Terms Used in Reviewed Articles**
1-1	Remote Rendering	“Streaming Architecture”
11	Single Software	NA
1-n	Multi-Client Remote Rendering	NA
1n	Single Software Multiple View	“Parallel Rendering Architecture”
1-nn	Client-Server	“Centralized Server”, “Server-Client”
1-n-n	Remote Rendering Server	“Client-Server”
1-nn-	Remote/Local Rendering Client-Server	“Client-Server”

1-n(-)n	Client-Server with Client Communication	"Hybrid", "Peer-to-Peer with Central Arbiter (PP-CA)"
n(-)n	Peer-to-Peer	"Peer-to-Peer", "Hybrid", "Federated Peer-to-Peer", "Hybrid P2P"
n-nn	Multi-Server Partitioning	"Multiple-Server", "Peer-to-Peer", "Distributed", "Clustered-Server"
n(-)-nn	Mirrored Multi-Server	"Proxy-Server Topology", "Tree-Based Server-Middleman-Client (TB-SMC)", "Peer-to-Peer", "Hybrid Peer-to-Peer", "Proxy Architecture", "Communication Proxy Based"
n-n(-)n	Multi-Server Partitioning with Client Communication	"Mirrored-Arbiter (MA)", "Hybrid Peer-to-Peer"
n(-)-n(-)n	Mirrored Multi-Server with Client Communication	"Enhanced Mirrored-Server"

We see that some terms such as "Hybrid", "Client-Server", and "Peer-to-Peer" are frequently used although they do not distinguish between architectures with important different characteristics. E.g., "Client-Server" (including "Server-Client") is used to label various architectures spanning over 3 GWG categories from classical client-server architectures to remote rendering architectures. Other terms such as "TB-SMC" contain too detailed information about *one* architecture, so that they are restricted to one or very few architectures - thus become useless for classification purposes. Finally, some terms like "Hybrid" are not sufficiently informative thus readers cannot understand the major characteristics of the architectures when the terms are used. The weaknesses of the traditional terminology for classifying game architectures highlight the contribution of the GWG framework both as a classification tool and for defining a more expressive and concise terminology. The remaining subsections discuss the game architectures in the review in detail in sequence according to the order of GWG categories as given in Table 1.

4.1 Graph Style *101* Architectures

The *101* category describes architectures consisting of one GW, and one PW. This does not necessarily indicate a single player game, but then the players must be located at the same place in order to interact with the single PW. *101* architectures can hardly be found in literature. One possible explanation is that this Graph Style usually represents traditional and trivial architectures, and thus not very interesting to researchers.

When a single GW and a single PW are natively connected, it belongs to *11* Flavor, which we have named Single Software architectures. Such architectures integrate all gaming processes and data into a single piece of software, where the simulation (implied by the GW) and the presentation (implied by the PW) are the two major computational tasks. The presentation of a game is directly generated from the game state simulated within the same running instance of the

game. Although there is no network involved, the architectures still can be complicated, because the simulation might involve techniques like AI, physics, and presentation might involve techniques like texture mapping and lighting that are difficult to carry out in real time. Game engines that implement such techniques are widely used in commercial game development and fit within the *11* Flavor. Although the history of Single Software architectures is as long as the history of the computer games, today such architectures are still widely used in popular video games such as New Super Mario Bros Wii, Rayman Ravin Rabbits and etc. Sollenberger and Singh present an architecture for affective social games in [5], which is the only architecture we found in our review that can be fitted into this category.

Another *101* Flavor identified in the review is the *1-1*. The remote Connector Type between the GW and the PW implies data flow across OS processes (usually through network). *1-1* architectures split simulation and presentation/interaction, so the task of each computational node is simplified. Thin clients with only graphics or text-based rendering and input handling capabilities are supported by such architectures. The simplification of the client software also decouples the client from some of the modules of the game engine, making it easier to provide support for cross-platform games (portability). Powerful computers can be used as servers to simulate multiple game sessions on the same machine, which optimizes the resource utilization by allocating available resource for other game sessions. In [4], Nave et al. present an architecture falling into *1-1* Flavor. The architecture deploys simulation and rendering on a server, while streaming the rendering result to a thin client which is only responsible for the on-screen presentation and user inputs management.

4.2 Graph Style *10n* Architectures

Similar to *101* architectures, architectures that fit into the *10n* pattern are also widely used in commercial games but are rarely found in the literature. By extending a single PW to multiple PWs, *10n* architectures provide support for multiplayer, multi-view games. The simplest case of *10n* is the *1n* Flavor, where the GW and the PW are native connected implying a multiplayer game without the need for network connectivity. Racing games are games that commonly utilize *1n* architectures, e.g. Mario Kart and Ridge Racer. Such games run on a single console connected to a single screen (TV), while multiple players can play through a split-screen that provides a separate view for each player. The rendering has to be done multiple times in the split-screen mode, which often cause performance challenges in order to provide smooth frame-rates with the same graphical fidelity. In [7], Lages et al. present a parallel architecture for multiple view rendering on GPU clusters which is appropriate for creating *1n* game architectures. The main idea is to combine sampler nodes with image-based renderers so some processes can be reused among the views. The method they propose can also be used for creating other architectures when it is integrated with different architectural patterns. For instance, if the parallel rendering is used with the remote rendering, the setup will describe a kind of architectures falling into the 1-n Flavor, which is another Flavor of the 10n Graph Style. Similar to *1-1* architectures, *1-n* architectures also distribute simulation and presentation to different nodes. But the difference is that multiplayer games with player-specific views are supported through multiple PWs. In [6], Bangun and Beadle

propose an architecture splitting the simulation and rendering processes of multiplayer games. The approach uses CPU servers to simulate the game state, and game consoles (e.g. Sony PlayStation, Nintendo 64) to render the game scenes and capture user input. The CPU server generates 3D/2D data related to the game state, and sends it to the game consoles. The architecture uses the simulation capability of CPU servers and the 3D/2D processing capability of game consoles and thus achieves an efficient resource usage. Streaming 3D/2D data also decouples game clients from many modules of game engine, which makes it easy to use heterogeneous devices.

1-n architectures use a single GW to maintain a consistent Game World, and multiple PWs to support more than one player in the game. Advantages and challenges of *1-n* architectures are similar to above-mentioned *1-1* architectures. Support of multiplayer games with player-specific views is the major improvement compared to the *1-1* architectures.

4.3 Graph Style *1nn* Architectures

The *1nn* Graph Style features a single GW connected to multiple LWs and PWs. Our review shows a great diversity of architectures falling into this category, resulting in 4 Flavors/sub-categories. All these Flavors use a remote Connector Type to connect the GW and the LWs, meaning that they all represent network game architectures. The analysis of the architectures revealed the following three differences between the Flavors:

1) The LWs and the related PWs could be simulated on the same node or not. (*1-nn* vs. *1-n-n*)
2) The LWs could be inter-connected or not. (*1-nn* vs. *1-n(-)n*)
3) The GW and the related PWs could be connected or not. (*1-nn* vs. *1-nn-*)

Among the four Flavors, *1-nn* is found to be the most popular one, and other Flavors can be seen as extensions of the *1-nn* Flavor in different ways.

4.3.1 Flavor 1-nn Architectures

The *1-nn* Flavor denotes that the LWs and the PWs are natively connected, meaning that they are simulated on the same node. Since the PW is the Game World that a player directly interacts with, the node is usually the computer of the player. The single GW is remotely connected with the LWs, meaning that the single GW is simulated on a node other than the players' machine, which is usually a dedicated server. In the reviewed articles, three architectures fall into the *1-nn* category. By analyzing the GWG Flavor, the following attributes of the architectures can be identified:

1) The player's machine maintains the local game state, rendering the game scene locally. The architectures with local rendering use the player's resources for rendering computation thus lessen the server capability requirements.
2) The GW is maintained on a dedicated server, which the players do not have direct access to. This makes such architectures better protected against cheating.
3) The single GW implies the risk of single point of failure.

One example of the *1-nn* Flavor is an architecture proposed by Caltagirone et al. for a massively multiplayer online role playing game engine [8]. Typical Client-Server pattern is used in the engine architecture and the layered system pattern is also used. The authors introduce the basic architecture with a single server and they also are aware of the scalability issues connected to a single central server, so they also discuss a possible extension of adding more servers to the basic architecture. Another example is Strifeshadow Fantasy (SSF) [9], a massive multiplayer online role-playing game with over 10,000 registered players. In SSF, a central server hosts the SSF server program simulating the entire Game World. Players connect to the server through the SSF client applications. HTTP is used for client-server interaction, and Macromedia Flash 4.0 is used to render the game scene. Clients regularly send Update Request Messages (URM) to the server, which is based on Active Server Page (ASP) combining the use of HTML, scripts, and Microsoft ActiveX server components. The server also uses a relational database for storing and retrieving the game state. Similar to SSF, SharedWeb [10] is an architecture for a 3D War Game over WWW. HTTP is also used in SharedWeb as the primary communication protocol between the client and the server, and since it is designed for 3D War Game and war simulation, 3D rendering is necessary. SharedWeb uses a specific web browser instead of Macromedia Flash on client to do 3D rendering, and the web browser supports and coordinates 3D rendering, Internet Relay Protocol based chatting, and HTML documents.

By examining the three architectures falling into *1-nn* Flavor, we can further summarize the characteristics of the architectures. *1-nn* is a straight forward and easy-to-implement solution for networked multiplayer games. The single GW in *1-nn* architectures is usually simulated on a server with high computational power and network capability, which enables multiple players to share a relatively complex Game World. Player resources are used to do rendering, which releases the server from this resource-consuming task. The centralized server makes it easier to do cheating-prevention and charging for services. Single point of failure is a major disadvantage of the *1-nn* architectures. The single server bottleneck also prevents *1-nn* architectures from being suitable for modern MMOGs with very complex scenes and a large amount of players. However, the *1-nn* architectures is one step in the right direction for supporting MMOGs.

4.3.2 Flavor 1-n-n Architectures

The *1-n-n* Flavor has also a single GW, but unlike *1-nn* architectures, the LWs and PWs are remotely connected. This means the presentation and management of related local states are distributed to two nodes.

In [11], Quax et al. present a Remote Rendering Server architecture for Networked Virtual Environments (NVEs), which can be the bases for computer games, but can also be used for other purposes. Quax et al. use rendering servers that are in charge of maintaining the local state, rendering and encoding rendering result to video streams which are played by thin clients such as mobile phones. A central world server is used to simulate the GW as in the other *1nn* architectures.

4.3.3 Flavor 1-n(-)n Architectures

The difference between the *1-n(-)n* and *1-nn* Flavor is that the former adds a remote Connector Type between LWs. The connectors can be used for exchanging local states, which reduces client latency and lessens the requirement of server bandwidth. This is because the direct client-to-client communication removes the need for all communication to go through the server.

The solution proposed in [13] is an example of the *1-n(-)n* architectures. Events sent by players are divided into two kinds: state-changing event (e.g. pick up a coin) and non-state-changing event (e.g. player movement). The state-changing events are sent to a central server owned by the game operator, and the non-state-changing events are sent to a "regional server", which is a player's machine in charge of forwarding events to players in that region. Note the difference between this "regional server" and the regional server in Multi-Server Partitioning architectures: for the latter, the regional servers simulate GWs. With regional servers, the architecture utilizes network bandwidth of the players, and sends non-state-changing events only to those relevant in the region to save the overall network load. The architecture improves the scalability and performance quality attributes of typical Client-Server architectures, but whether an event is a "state-changing event" or not, might be highly dependent on the actual game design and is therefore hard to determine.

Another example of the *1-n(-)n* architectural pattern is the approach proposed by Pellegrino and Dovrolis in [14]. This is a hybrid architecture named Peer-to-Peer with Central Arbiter (PP-CA), which modifies Peer-to-Peer architecture by adding a central arbiter, resulting in a simple consistency resolution and scalability at the same time. Games based on this architecture run on player's client devices as well as on a central server (arbiter). Player clients exchange updates by direct communication. Whenever a player client sends an update, it also sends an update to the central arbiter, and the arbiter will check for state inconsistencies, and correct the player clients if inconsistency is detected in their local state.

4.3.4 Flavor 1-nn- Architectures

The remote Connector Type between the GW and PWs is the main characteristic of *1-nn-* Flavor, meaning that *some* of the PWs are generated on the same node where the GW is simulated, while *others* are generated on the nodes simulating the LWs. Architectures that belong to this GWG Flavor support both client and server rendering.

Only one of the architectures from our review falls into *1-nn-* Flavor, which is the Pervasive Multiplayer Multiplatform Game (PM2G) architecture [12]. In PM2G, players are mostly connected to the virtual world, but they may also play against each other using heterogeneous devices. To enable PM2G, Trinta et al. propose a PM2G architecture employing the client-server pattern as well as the layered system pattern. The PM2G architecture includes three layers: the PM2G server, the PM2G services, and the PM2G client application. The architecture supports user-context management and content/interaction adaptation through the PM2G services

layer. *Context* includes relevant information about the player's connection to the virtual world, such as his or her device and preferences. The context is used by the PM2G architecture to provide the *content adaptation* service, which is a process to translate game state to a proper data representation according to the player context. Players using thin clients may receive the text-based game state, while players using more powerful clients may receive the default game state and their devices will do the rendering and presentation. The text-based game state is actually the rendering result, which is produced by the server through translating and serializing of a part of the GW that is within the area of interest of a specific player. Streaming text-based game state is similar to streaming 3D data or video in the computation distribution aspect, but requires much lower bandwidth and simpler player clients.

Supporting both client and remote rendering makes the *1-nn-* architectures more flexible by enabling more heterogeneous player clients compared to *1-nn* architectures. However, the approach that only supports text-based remote rendering has limited usefulness. An approach that provides both local and remote graphical rendering can be a topic for further research.

4.4 Graph Style *n0n* Architectures

The *n0n* Graph Style features multiple GWs and PWs, where the PWs are directly connected to GWs without LWs. This means that the game externalization does not rely on a local state. Possible Flavors of the *n0n* Graph Style include the *n-n*, the *n(-)-n*, and the *n(-)n*. However, only the *n(-)n* Flavor was found in the articles included in our review.

The *n(-)n* Flavor features a native Connector Type between the GWs and the PWs, and a remote Connector Type between the GWs. The PW is the Game World players directly interact with, so it is always simulated on the player's machine. Considering that the GW and the PW are natively connected, the GW is simulated on the same player's machine. Since all Game Worlds are simulated on players' machines, we name such architectures Peer-to-Peer architectures. Note however, that the *n(-)n* Flavor is not the same as the Peer-to-Peer architectural pattern, because the GWG framework classifies game architectures from the *Game World distribution* perspective, which is different from the *network connection* perspective. In traditional taxonomies, when the server is running on some players' machines, the architecture will be also called "Peer-to-Peer architecture" from the network connection's perspective in some articles and "public server architecture" from the responsibility distribution's perspective in some other articles. With the GWG framework, this kind of architecture is classified into the *nnn* Graph Style but not the *n(-)n* Flavor that we are discussing here, because player machines that run servers are considered as GW hosts and player machines running pure clients are considered as LW and PW hosts.

By analyzing the characteristics of this Flavor, we have found the following advantages and challenges of *n(-)n* architectures.

Advantages:

1) Using player resources to simulate all Game Worlds can provide better scalability;
2) Multiple GWs are simulated, and it can potentially avoid single point of failure;

Challenges:

1) Because the GWs are simulated on player machines and are inter-connected, maintaining state consistency and persistency becomes difficult;
2) Players can directly access the GW, which brings challenges such as anti-cheating.
3) For some MMOGs with complex game worlds, simulation of the whole GW demands a lot of resources from the player machines, making it a challenge to implement *interest management* that limits the objects account on a node.
4) Each player machine is responsible for sending updates of the GW it is simulating to other clients. This causes corresponding network traffic, which requires optimization.

The works described in the reviewed articles are mainly focused on dealing with above challenges.

Due to the scalability issues of network level multicast protocols, Application Level Multicast (ALM) is practically adopted in some architectures to optimize network traffic. A peer-to-peer game architecture with a tree-based multicast approach is proposed in [15], where player nodes are connected to each other forming a multicast tree. The nodes that are found to minimize latency from all players will be selected as the "center" of the tree. We also found another example of tree based ALM in [16]. Interest management is usually managed by ALM due to the requirement to limit network traffic. Rooney et al. introduce a peer-to-peer architecture with ALM in [17], which use interest management to organize nodes according to their area of interest. Each area of interest may be connected with a "multicast reflector", which is responsible for forwarding messages sent by nodes in the area of interest to all subscribers.

To maintain a consistent global state among nodes, various protocols are proposed. One solution is VoroGame [19]. A structured P2P overlay based on a distributed hash table (DHT) (e.g. [44,45]) combined with an unstructured P2P overlay based on a Voronoi diagram [46] is used to manage the global state. The Game World is divided into zones based on a Voronoi diagram, and DHT is used to store game objects on peers evenly and fairly. Each peer has two roles in the architecture 1) Voronoi responsible related to a zone in the Voronoi diagram, which is in charge of simulate behavior of objects inside the zone that the peer locates in, and 2) DHT responsible that is in charge of storing and forwarding object state changes from Voronoi responsible to players that are interested in the objects state. The use of a "super node" is a common solution to maintain state consistency, and the super node can solve the inconsistency by providing the final decision of the global state or part of the global state. An example is a hybrid architecture that is proposed in [16], where the Game World is divided into hexagonal zones. Each zone has a super node in charge of coordinating the interaction among players in that region. Super nodes can also be used to detect cheating because a super node could recognize a manipulated game state. An important benefit of the super node approach is saving service cost by using player resources. Mediator [20] is a p2p architecture using resources provided voluntarily by the player. Four kinds of super nodes (mediators) are introduced and a player can announce their resource (computation power, bandwidth, etc.) and be assigned tasks by the super nodes. They can also be promoted to super nodes when needed. A reward scheme according to player contribution is also discussed in [20]. The super node can be a security threat since the system does not have complete

control of the node and the super node and player client can be the same machine. ACORN [21] is a Peer to Peer architecture with dynamic coordinator (super node). The coordinator could be moved from one node to another in order to make cheating hard if not impossible. Coordinator Access Point (CAP) is introduced to help peers to find the coordinator. In DACA [22], super nodes of regions are selected from the player nodes whose avatars are not in the corresponding regions. In [23], coordinators (super nodes) which are in charge of objects states in a region are randomly selected. The objects can be divided into subsets based on types, and the objects of each type are managed by a coordinator. The coordinator is also the region server for a region. Interest management and multicast are also discussed in [23]. Players in each region form an interest group, and transient state updates of a player will be multi-casted to the group, and if the game mechanics allows, to the players in another group who have subscribed to these state changes. Besides the state consistency, state persistency is also a research direction. Hampel et al. present a Peer-to-Peer MMOG architecture in [18], where Pastry is used to implement the P2P network infrastructure, and its extensions, Past and Scribe, are used to implement persistent object storage and event distribution.

The literature review reveals two challenges of *n(-)n* architectures apart from those addressed in the GWG reasoning:

1) It is difficult to charge players for gaming services when players do not connect to any centralized server managed by the operators.
2) A reward mechanism must be applied to encourage players to contribute with their computational and network resources.

It is possible to foresee several more *n0n* graph Flavors, however, all architectures in our review fall into the *n(-)n* Flavor. This may indicate a potential research direction of exploring un-implemented *n0n* Flavors. It is also worth mentioning that *n0n* architectures are mainly proposed by researchers and are rarely (if not never) used in commercial games.

4.5 Graph Style *nnn* Architectures

nnn architectures attract most interests from researchers. Multiple GWs means that the global state is distributed or duplicated, while multiple LWs with PWs means that multiple instances of local state are maintained with multiple player views. We found 4 different Flavors in literature, which are *n-nn*, *n(-)-nn*, *n-n(-)n,* and *n(-)-n(-)n*. The major differences between these Flavors are:

1) The GWs are connected with each other or not: *n-nn* and *n-n(-)n* vs. *n(-)-nn* and *n(-)-n(-)n*
2) The LWs are connected with each other or not: *n-nn* and *n(-)-nn vs. n-n(-)n* and *n(-)-n(-)n*

4.5.1 Flavor n-nn Architectures

When multiple GWs are not interconnected but remotely connected to LWs, *nnn* architectures fall into the *n-nn* Flavor. The disconnected nature of the GWs normally means they do not share a common state, so there is no need for synchronizing the GWs. Thus, the entire global game state is

partitioned. Partitioning the Game World can reduce computation load and network traffic demands on one single server, and thus make the architecture more scalable.

An intuitive partitioning approach is spatial partitioning, i.e., dividing a single Game World into zones based on virtual spatial distribution. We name such architectures *Zone-Based Architectures*. One example is presented in [24], where the virtual world are divided into regions. Each region is a disjoint piece of the GW and it can be any convex polygon. Each region is assigned to a server that is responsible for simulating this piece of the GW. Each player client will only exchange messages to a server simulating the piece of Game World holding his/her avatar. The spatial partitioning could also be dynamic instead of predefined. Real Tournament (RT) [25] is an augmented reality FPS game providing a multiplayer mode. Each player joining the game will be assigned to a zone of the Game World, and will then be in charge of the objects' states of that zone. The spatial partitioning is dynamic, meaning that at the beginning, the first player is in charge of the whole world, and when a second player joins, the whole world will be divided into two zones and assigned to two players respectively. As more players join, the zones will be further sub-divided. This architecture utilizes the characteristics of location-aware games that the proximity of avatars implies proximity of players in the real playgrounds, and thus achieves both the network optimization and the computation load balance. Another dynamic partitioning architecture is proposed in [26]. The Game World is divided vertically against the x-axis in the architecture, and a server node manages each partition. If a node is overloaded, it can assign some game units (object or characters) to its neighbor by resizing the partition it is processing. Both of the approaches keep the spatial *continuity* of the partition on each server, when the system dynamically adjusts the partitions. But it is not always necessary. The Game World could be further divided into microcells [27], and a group of microcells forms a zone dynamically. The microcells belonging to the same zone do not have to be adjacent, thus non-spatially-continuous partitioning is supported. The microcell concept is proposed by Vleeschauwer et al. in [27] for an architecture that balances processing load among multiple servers. One server manages one or many microcells to achieve a balanced server allocation. A similar approach is introduced in [28], where Hori et al. divide game space into grids and the grids are dynamically assigned to regions. An alternative approach to spatial partitioning is object-based partitioning. This approach detaches the relationship between virtual location and game objects, and assigns game objects to servers based on other object attributes. Hsu et al. propose a Clustered-Server architecture in [29], where a single Game World can be distributed to a group of connected servers. Their work does not focus on a specific partitioning approach, but presents some possibilities of the world allocation, e.g. region oriented, and (virtual objects) hierarchically structured.

Another research topic of the *n-nn* Flavor is how to "efficiently" assign partitions to servers. The classical solution is to assign partitions according to server load to optimize utilization of server capability. However, such approaches disregard the player context in the real world. E.g. a player may be assigned to a server that is far away from his physical location. An interesting solution is proposed in [30], which describes an architecture with geographical distributed proxies (zone servers) to minimize the latency due to geographical distance. The proxy to be selected for a zone is decided by the overall geographical proximity of the players in that

zone and the proxies. Another efficient way to assign zones to servers is based on players' activities in the zone. Distributed Entertainment Environment (DEE) [31] is an online game architecture developed at BT laboratories. DEE divides the Game World into predefined partitions, and a process will manage each partition. If the state of a zone becomes completely predictable over time, its state is written back into a persistent object store, and the relevant process is closed. If a new activity occurs within a zone, a new process is started and the previous zone state is restored.

Most of the above solutions use dedicated servers controlled by a game service provider. However an attractive alternative can be to use players' computer resources to reduce service costs (public servers). In [32], the authors discuss the dynamic partitioning of a Game World as the player number increases, and the usage of public servers to simulate a zone in the game space. Similarly, a public server architecture is introduced in [33]. In this architecture some player nodes will be used as Regional Controllers (RC). The GW is divided into regions, which are simulated on RCs. Instead of the conventional approach of putting one server in charge of one or more regions, this architecture uses a group of RCs to simulate one region. The game state of the region on each RC in the group should be the same. The RCs do not communicate with each other to keep consistent game state as the consistency depends on a voting system running on the clients, which should also keep the system cheat-proof.

The *n-nn* architectures use multiple instances of GWs to distribute simulation to multiple nodes. The multiple isolated GWs reduce the computational load and reduce network traffic on a single server without introducing consistency problems. The major advantages of *n-nn* architectures are the better scalability than *1nn* architectures, and simpler consistency control than *n(-)-nn* architectures (we will discuss this issue later in the article). The major challenge of *n-nn* architectures is that as they simulate the Game World on different nodes, the players may have to connect to a node that is not network-wise close to the player.

4.5.2 Flavor n(-)-nn Architectures

The remote Connector Type between multiple GWs differentiates *n(-)-nn* from *n-nn* architectures. The interconnections between GWs require them to be synchronized, which implies that the GWs are either identical or sharing common parts.

We name architectures with identical interconnected GWs as Mirrored Multi-Server Architectures. Rokkatan [34] is a multiplayer online game based on this kind of architectures. The game state is simulated on multiple servers, and each server manages a full instance of the entire game state. The servers use a cooperation protocol to keep the game state consistent. A player joining the game can select the fastest server to get the smoothest gaming experience. Another example is the Tree-Based Server-Middlemen-Client (TB-SMC) architecture proposed by Matuszek [47]. A description of how the TB-SMC architecture is evaluated in terms of performance by developing a simulator can be found in [35]. TB-SMC architecture uses two servers that simulate the same Game World. One is the primary server in charge of updating all clients, and the other is the backup server that is used to provide continuous service when primary

server fails. The middlemen are nodes that are used to forward messages from the server to the clients and to send messages collected from the clients to the server. The nodes are connected in a tree, and the number of middlemen as well as the depth of the tree can be adjusted when number of players increases or decreases. A double central server used in TB-SMC architecture also reduces the risk of single point of failure. Hydra [36] proposed by Chan et al. is a similar solution using a double central server, but in this approach the player machines are also used to simulate GWs. Each player node acts as both client and server. The Game World is divided into zones and each zone is called a "slice". Hydra replicates slices with multiple copies, and each node can be in charge of one or more slices. Each zone has a primary slice simulated by a server, which keeps the state up to date according to player client messages. The backup slices servers are responsible for updating backup slices. Another kind of *n(-)-nn* architectures features the combination of the GW duplication and the GW partition. One example is the public server architecture proposed in [37], where a central server is used to process log-in and manage sub-servers. The sub-servers are selected from player nodes. Each sub-server is assigned with a partition of the GW and thus becomes the *partition responsible*. Each sub-server also maintains a copy of each adjacent partition managed by other sub servers. A partition is simulated by its partition responsible, and its copies maintained on other sub-servers are updated accordingly through network communication. When a server fails (e.g. a player leaving the game), the central server could recover partition state using copies maintained by other sub servers, and the central server will take charge of the partition for a period until a new sub server is selected. The partitions can also be sub-divided and area servers are introduced to manage the sub partitions, which are also player clients.
In n(-)-nn architectures with non-identical GWs, servers simulating different GWs may have different purpose or authority. E.g. a Generic proxy architecture is proposed in [38]. A central server and multiple proxies are used to coordinate the game state updates. The server has the final authority, and the proxies are trusted to some extent because players do not control them. Proxies act as an extension of the central server and players can choose to interact through the proxies. Another proxy architecture is proposed in [39], which uses geographically distributed proxies to handle "time critical" computations like simulation of player movement. Here, the players connect to a nearby proxy, and send input events to the proxy. The proxy will simulate the players' movement, send a new state to the players, and multicast the state to other proxies if necessary. The global game state except the time critical state information will still be handled by a central server, or by multiple servers if the world is divided into zones.

The advantages of *n(-)-nn* architectures are geographical optimization of network communication as well as avoiding a single point of failure. The major challenge of *n(-)-nn* architectures is how to develop a protocol that insures consistency.

4.5.3 Flavor n-n(-)n Architectures

The *n-n(-)n* Flavor represents an extension to the *n-nn* architectures by adding a remote Connector Type between LWs. This means that the local states of the players can be exchanged. This approach is usually used to improve performance. By enabling direct player-to-player interaction,

client latency will be reduced compared to traditional communication protocols that have to go through a server. Due to the direct connection between clients, the *n-n(-)n* architectures are often named Peer-to-Peer architectures in traditional taxonomies. However, the GWG framework has shown that the Peer-to-Peer architectural pattern can represent quite different approaches, which are represented in multiple Graph Styles and Flavors. So we name the *n-n(-)n* architectures Multi-Server Partitioning with Client Communication.

ALVIC-NG [40] is a proxy-based architecture, which uses central servers to simulate the game state, and a group of proxies to reduce communication load on the servers as well as on the clients. The proxies act as mediators for server-client communication and are not in charge of the computation of the game state. The game state on the proxies can also serve as cached data in the system to reduce latency in some cases, so it can be mapped to the LWs in the GWG framework. The architecture divides the world into regions managed by logical servers. The regions can be split or reassigned to servers that can dynamically be added. Similarly, Mirrored Arbiter (MA) [41] is an architecture using multiple arbiters (servers). The Game World is partitioned into regions. Each arbiter is responsible for state computation and ensures consistency of a region. Clients are fully connected and communicate directly to exchange states. Clients are also kept connected to the arbiter in order to deal with state consistency. HERA [42] is a public server architecture which groups peers into servers and clients. The servers are in charge of managing the game objects. Clients within the same Area Of Interest (AOI) are organized as a fully connected graph in order to directly communicate with each other to increase interactivity.

The *n-n(-)n* architectures enable the exchange of the local game state without going through the servers and thus decrease latency. However, a problem related to such architectures is security, i.e. the game state exchanged must not be misused in order to break the balance or gain improper advantages over others players.

4.5.4 Flavor n(-)-n(-)n Architectures

Intuitively, the *n(-)-n(-)n* architectures extend *n(-)-nn* architectures through allowing direct communication between player clients to reduce network latency and network load. The Enhanced Mirrored Server (EMS) [43] architecture is one example of this Flavor. The peers (clients) in EMS can communicate directly to exchange game state updates, and multiple mirrored servers are used to reduce latency. The connection between clients reduces network latency and also reduces network and computation load on the server. Like the *n(-)-nn* Flavor, the benefit of simulating the entire game state using multiple servers comes with the challenge of synchronizing the game state.

The *nnn* architectures are leading-edge techniques that attract the most of interests from researchers in this field. Multiple LWs and PWs implies multiplayer mode, and multiple GWs can potentially reduce players as well as objects density on a single node thus reduce computation and network load. The key characteristic of *nnn* architectures is the use of multiple GWs. We have identified three main challenges related to such architectures:

1) It is difficult to keep consistency among GW instances when they are inter-connected;
2) Single point of failure when GWs are isolated; and
3) Difficult to implement.

5 Discussion

Game architectures have kept evolving ever since the first computer game was developed. The roadmap started with *11* architectures providing a single game software with a single view on the same machine. Although it is relatively simple, it is still the most appropriate solution for most single player commercial games today. An extension is multiplayer game based on the *11* architectures where players share a single view, which is natural and convenient for some game genres (e.g. 2D action games and shooting games), but inappropriate for games that require individual player views (e.g. racing games). To support the latter, the *1n* architectures must be used which supports multiple player views (PWs) rendered by a single program. The *11* and *1n* architectures have dominated the game industry for ages before the *1-nn* architectures opened the door for online games. The *1-nn* architectures provide sharable and consistent Game Worlds, centralized control, and less chances of cheating. However, the 1-nn architectures are limited in terms of scalability, which prevents them from being used in MMOGs.

Work addressing the scalability issue of the *1-nn* architectures has been done in two directions: 1) use more servers to share computation load, and 2) use player resources to reduce server load. The *first approach* is represented by the *n-nn* architectures, where the GW is divided into predefined pieces (e.g. regions), and players in different places of the Game World may be assigned to different servers. Partitioning of the Game World is the major characteristic of this kind of architectures, thus named Multi-Server Partitioning architectures. Spatial partitioning is the most popular solution to divide the game world, but object-based partition has also been explored. World partitioning can further be extended to dynamic partitioning, where strategies such as resizing and Microcell are applied to achieve better load balancing.

The *second approach* is represented by the *n(-)n* and *n-nn* architectures. The former is named Peer-to-Peer architectures where the GW is simulated on players' machines, and where the main challenge is to keep the global game state consistent. The latter is named Public Server architecture where selected player machines serve as the hosts for simulating the GW, while clients connect to the hosts just as they do in the Multiple-Server Partitioning architectures. The possible problems related to the use of player resources are issues related to security and availability, which motivate research on such architectures.

Both *1-nn* and *n-nn* architectures may cause high latency to some players in MMOGs, since players are distributed over the world and the server they are connected to may not be geographically or network-wise close. A solution to this issue is the use of proxies (mirrored servers), which host the same GW but are distributed to different geographical regions. This solution is implemented in *n(-)-nn* architectures, where the remote Connector Type between GWs implies that the state synchronization is carried out among mirrored servers. Besides the latency issue, single point of failure is also eliminated when using *n(-)-nn* architectures.

Above-mentioned architectures represent the trunk of the game architecture roadmap, and there are also some branches connected to two major architectural strategies. One strategy is Remote Rendering. When thin-clients such as set-top boxes and low-end cellphones which are not capable of real-time graphical rendering are used in a graphical game, server rendering is necessary. The graphics or image based rendering results can then be streamed to the thin-clients over the network. The strategy can be combined with many GWG Flavors: when Remote Rendering is used in combination with the *11* architectures, it will result in the *1-1* architectures. When Remote Rendering is used in combination with the *1-nn* architectures, it will result in the *1-n-n* architectures. This strategy can be open for further exploration, as potential architectures such as *n-n-n* have not yet been proposed. Another strategy is Client Communication. When the GW and the LW are distributed to two kinds of computational nodes, the synchronization of the LWs is usually done through a LW- GW- LW route, which brings additional latency. Allowing direct state exchange among LWs can solve this problem. Combing the Client Communication strategy with various GWG Flavors results in corresponding extensions: when Client Communication is combined with the *1-nn* architectures it will result in *1-n(-)n* architectures, and when Client Communication is combined with the *n-nn* and *n(-)-nn* architectures, it will result in *n-n(-)n* and *n(-)-n(-)n* architectures.

Architectures are developed for fulfilling specific functional and non-functional requirements. So it makes no sense to say which architecture is the "best" in general, but for a given problem domain, there may be one or more architectures which are more appropriate than others. Table 3 shows which game types each GWG Flavor is usually applied to, i.e. for a game type which architecture sub-category(s) is appropriate in practice.

Table 3 GWG Flavors and The Corresponding Game Types

GWG Flavor	**Game Type**
1-1	Single-Player Game on Thin Client
11	Typical Single-Player Game, Multiplayer Single-Screen Game
1-n	Multiplayer Online Game on Thin Client
1n	Multiplayer Split-Screen Game
1-nn	Multiplayer Online Game
1-n-n	Multiplayer Online Game on Thin Client
1-nn-	Multiplayer Online Game with Thin Client and Standard Client
1-n(-)n	Multiplayer Online Game
n(-)n	Multiplayer Online Game, MMOG
n-nn	MMOG
n(-)-nn	
n-n(-)n	
n(-)-n(-)n	

As we showed in [48], the GWG framework can also be used for exploring future architectures. The basic idea is that if we create a list of all possible Graph Styles and Flavors, and fill the list with the data gained from the systematic review, we can find some "holes" (unused Graph Styles and Flavors) in the list. Each unused Graph Style or Flavor may imply an unknown game architectural pattern, which can serve as a base for exploring unknown architectures. Further, we have presented an example of creating a Hybrid Remote Rendering Architecture with the *1nn* Graph Style in [48] to validate the method. The architecture is appropriate for mobile and ubiquitous games, and it has not been found in any existing article in the systematic review. Fitting existing architectures into the GWG framework is a necessary step to exploring new architectures with the framework, so it is also an important contribution of this article.

6 Conclusion

The game architectures are getting more and more complicated, and thus attracting increasing interests both from researchers and practitioners. Our systematic review summarizes studies and findings on game architectures in the literature. Through the review, we have proven the usefulness of the GWG framework as an architectural taxonomy.

From Single Software to Remote Rendering architectures, from Multi-Server to Peer-to-Peer architectures, this article has drawn a big picture of game architecture research and technologies. Game architectures that do not include network are rarely described in literature, although these are the popular architectures in commercial single player games or multiplayer games running on one machine (a PC or a console). Online game architectures, especially MMOG architectures attract most of the interests from the researchers. Multi-server architectures (*nnn* Graph Style) and Peer-to-Peer architectures (*n0n* Graph Style) represent the two major architectural patterns for MMOG in game architecture research. For Multi-server architectures, partitioning of the global Game World and state consistency are issues being widely discussed. For Peer-to-Peer architectures, more design issues are addressed by the studies in literature (see [2] for more details).

The architectures we have reviewed may feature one or more architectural strategies, e.g., zoning, use of player resources, etc. These tactics are constructed to handle particular architectural drivers such as reducing server computational load and anti-cheating. This article has described and discussed these tactics and drivers. It will be further synthesized in future work. The systematic review only includes architectures described in the literature, and there are also a lot of architectures of popular commercial games without any public available documentation. Applying the GWG framework in analyzing these architectures may produce more interesting results. However, it is difficult to gain the access to such architectures, especially for games that are not open source. We are aware of this limitation in our review, but we hope to review such unpublished architectures in the future.

Acknowledgements

We want to thank the anonymous reviewers for their many suggestions that improve the content much.

References

1. Xinbo J, Safaei F, Boustead P Latency and scalability: a survey of issues and techniques for supporting networked games. In: Networks, 2005. Jointly held with the 2005 IEEE 7th Malaysia International Conference on Communication
2. Fan L, Trinder P, Taylor H (2010) Design issues for Peer-to-Peer Massively Multiplayer Online Games. Int J Adv Media Commun 4 (2):108-125.
3. Dybå T, Dingsøyr T (2008) Empirical studies of agile software development: A systematic review. Information and Software Technology 50 (9-10):833-859
4. Nave I, David H, Shani A, Tzruya Y, Laikari A, Eisert P, Fechteler P Games@large graphics streaming architecture. In: Consumer Electronics, 2008.
5. Sollenberger DJ, Singh MP Architecture for affective social games. In, Tiergartenstrasse 17, Heidelberg, D-69121, Germany, 2009. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Springer Verlag, pp 79-94
6. Bangun RA, Beadle HWP A network architecture for multiuser networked games on demand. In: Information, Communications and Signal Processing, 1997
7. Lages W, Cordeiro C, Guedes D Performance analysis of a parallel multi-view rendering architecture using light fields. In, Tiergartenstrasse 17, Heidelberg, D-69121, Germany, 2009. Visual Computer. Springer Verlag, pp 947-958
8. Caltagirone S, Keys M, Schlief B, Willshire MJ (2002) Architecture for a massively multiplayer online role playing game engine. J Comput Small Coll 18 (2):105-116
9. Chan HT, Chang RKC Strifeshadow Fantasy: a massive multi-player online game. In: Consumer Communications and Networking Conference, 2004
10. Huang J-Y, Chang J-L, Li C-W, Lin K Design of a multiple participant 3D war game environment over WWW. In, Orlando, FL, United states, 1998. Proceedings of SPIE - The International Society for Optical Engineering. SPIE, pp 303-312
11. Quax P, Geuns B, Jehaes T, Lamotte W, Vansichem G On the Applicability of Remote Rendering of Networked Virtual Environments on Mobile Devices. In: Systems and Networks Communications, 2006
12. Trinta F, Pedrosa D, Ferraz C, Ramalho G (2008) Evaluating a middleware for crossmedia games. Computers in Entertainment (CIE) 6 (3)
13. Jardine J, Zappala D A hybrid architecture for massively multiplayer online games. In, Worcester, MA, United states, 2008. Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games, NetGames'08. Association for Computing Machinery, pp 60-65
14. Pellegrino JD, Dovrolis C (2003) Bandwidth requirement and state consistency in three multiplayer game architectures. Paper presented at the Proceedings of the 2nd workshop on Network and system support for games, Redwood City, California
15. Ramakrishna V, Robinson M, Eustice K, Reiher P An active self-optimizing multiplayer gaming architecture. In: Autonomic Computing Workshop, 2003
16. Ahmed DT, Shirmohammadi S, De Oliveira JC (2009) A hybrid P2P communications architecture for zonal MMOGs. Multimedia Tools and Applications 45 (1-3):313-345
17. Rooney S, Bauer D, Deydier R (2004) A federated peer-to-peer network game architecture. Communications Magazine, IEEE 42 (5):114-122
18. Hampel T, Bopp T, Hinn R (2006) A peer-to-peer architecture for massive multiplayer online games. Paper presented at the Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games, Singapore,
19. Buyukkaya E, Abdallah M, Cavagna R VoroGame: A Hybrid P2P Architecture for Massively Multiplayer Games. In: Consumer Communications and Networking Conference, 2009. CCNC 2009. 6th IEEE, 2009. pp 1-5
20. Fan L, Taylor H, Trinder P (2007) Mediator: a design framework for P2P MMOGs. Paper presented at the Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games, Melbourne, Australia,
21. Norden S, Guo K (2007) Support for resilient Peer-to-Peer gaming. Computer Networks 51 (14):4212-4233

22. Huey-Ing L, Bing-Rong T DACA: Dynamic Anti-Cheating Architecture for MMOGs. In: Advanced Information Networking and Applications, 2009
23. Knutsson B (2004) Peer-to-peer support for massively multiplayer games
IEEE INFOCOM 2004 INFCOM-04.
24. Assiotis M, Tzanov V (2006) A distributed architecture for MMORPG. Paper presented at the Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games, Singapore,
25. McCaffery DJ, Finney J (2004) The need for real time consistency management in P2P mobile gaming environments. Paper presented at the Proceedings of the 2004 ACM SIGCHI International Conference on Advances in computer entertainment technology, Singapore
26. Min D, Choi E, Lee D, Park B (1999) Load balancing algorithm for a distributed multimedia game server architecture. International Conference on Multimedia Computing and Systems - Proceedings 2:882-886
27. Vleeschauwer BD, Bossche BVD, Verdickt T, Turck FD, Dhoedt B, Demeester P (2005) Dynamic microcell assignment for massively multiplayer online gaming. Paper presented at the Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games, Hawthorne, NY
28. Hori M, Iseri T, Fujikawa K, Shimojo S, Miyahara H Scalability issues of dynamic space management for multiple-server networked virtual environments. In, Victoria, BC, Canada, 2001. IEEE Pacific RIM Conference on Communications, Computers, and Signal Processing - Proceedings. Institute of Electrical and Electronics Engineers Inc., pp 200-203
29. Hsu C-C, Ling J, Li Q, Kuo CCJ On the Design of Multiplayer Online Video Game Systems. In, Orlando, FL, United states, 2003. Proceedings of SPIE - The International Society for Optical Engineering. SPIE, pp 180-191
30. Beskow PB, Vik K-H, Halvorsen P, Griwodz C (2009) The partial migration of game state and dynamic server selection to reduce latency. Multimedia Tools and Applications 45 (1-3):83-107
31. Powers SJ, Hinds MR, Morphett J (1997) Distributed entertainment environment. British Telecom technology journal 15 (4):173-180
32. Merabti M, El Rhalibi A Peer-to-peer architecture and protocol for a massively multiplayer online game. In: Global Telecommunications Conference Workshops, 2004. GlobeCom Workshops 2004. IEEE, 2004. pp 519-528
33. Kabus P, Buchmann AP (2007) Design of a cheat-resistant P2P online gaming system. Paper presented at the Proceedings of the 2nd international conference on Digital interactive media in entertainment and arts, Perth, Australia
34. Muller J, Jan HM, Alexander P, Maraike S, Sergei G (2006) Rokkatan: scaling an RTS game design to the massively multiplayer realm. Comput Entertain 4 (3):11
35. Guo Y, Fujinoki H Tree-based server-middleman-client architecture: Improving scalability and reliability for voting-based network games in Ad-Hoc wireless networks. In, Boston, MA, United states, 2006
36. Chan L, Yong J, Bai J, Leong B, Tan R (2007) Hydra: a massively-multiplayer peer-to-peer architecture for the game developer. Paper presented at the Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games, Melbourne, Australia
37. Lee H-H, Sun C-H (2006) Load-balancing for peer-to-peer networked virtual environment. Paper presented at the Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games, Singapore
38. Mauve M, Fischer S, Jorg W (2002) A generic proxy system for networked computer games. Paper presented at the Proceedings of the 1st workshop on Network and system support for games, Braunschweig, Germany
39. Aggarwal S, Christofoli J, Mukherjee S, Rangarajan S (2006) Authority assignment in distributed multi-player proxy-based games. Paper presented at the Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games, Singapore
40. Quax P, Dierckx J, Cornelissen B, Vansichem G, Lamotte W Dynamic server allocation in a real-life deployable communications architecture for networked games. In, Worcester, MA, United states, 2008. Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games, NetGames'08. Association for Computing Machinery, pp 66-71
41. Yang L, Sutinrerk P (2007) Mirrored arbiter architecture: a network architecture for large scale multiplayer games. Paper presented at the Proceedings of the 2007 summer computer simulation conference, San Diego, California
42. Bettinger C, Baumann A, Hausen F, Schneider G, Schloss H HERA: Design Framework for Decentralized Distributed Virtual Environments and Games. In: INC, IMS and IDC, 2009

43. Webb SD, Soh S, Lau W (2007) Enhanced mirrored servers for network games. Paper presented at the Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games, Melbourne, Australia
44. Rowstron A, Druschel P (2001) Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In: Guerraoui R (ed) Middleware 2001, vol 2218. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, pp 329-350
45. Stoica I, Morris R, Karger D, Kaashoek MF, Balakrishnan H (2001) Chord: A scalable peer-to-peer lookup service for internet applications. SIGCOMM Comput Commun Rev 31 (4):149-160
46. Aurenhammer F (1991) Voronoi diagrams- a survey of a fundamental geometric data structure. ACM Comput Surv 23 (3):345-405
47. Matuszek S, Freelander J (1999) Othecko: A Distributed Voting-Based Game System. University of North Carolina at Chapel Hill
48. Zhu M, Wang AI, Guo, H, Trætteberg H (2012) Graph of Game Worlds: New Perspectives on Video Game Architectures, Manuscript submitted for publication