Graph of Game Worlds: New Perspectives on Video Game Architectures

ART#

MENG ZHU, ALF INGE WANG, HONG GUO, and HALLVARD TRÆTTEBERG
Department of Computer and Information Science,
Norwegian University of Science and Technology, Norway

Software architectures of video games have evolved radically over the last decade. Researchers and practitioners have proposed architectural patterns such as stand-alone configuration, Peer-to-Peer, Client Server and Hybrid, and some of them have successfully been used in commercial game titles. A conceptual framework for game architectures – Game Worlds Graph (GWG) – is presented in this paper for three purposes: 1) classifying existing architectures, 2) communicating and sense-making of game architectures, and 3) exploring and discovering future game architectures. The framework is based on the Game World and the World Connector concepts, which reveal some essential characteristics of game architectures – thus can be more descriptive and informative than existing taxonomies.
Categories and Subject Descriptors: K.8.0 [**Personal Computing**]: Games, D.2.11 [**Software Engineering**]: Software Architectures-*Domain Specific architectures*
General Terms: Taxonomy, Conceptual Framework
Additional Key Words and Phrases: Game Worlds Graph, Game World, World Connector

1. INTRODUCTION

Since the first popular commercial game Pong was released in 1972 [1], game developers have created video games with an increasing amount of content. Today, game software has a much higher complexity compared to games developed 30 years ago. The evolution of game software architectures from 1994 to 2004 was illustrated in [2], where a typical 2D game from 1994 consisted of five main modules compared to over 30 modules for typical massively multiplayer 3D game in 2004, where each module consisted of between six thousand to forty thousand lines of code. In line with the increasing game software size and complexity, the architecture patterns used by game developers have also evolved over time. Before the early 1990s, most games only consisted of a single application, where all computations were performed on the player's machine. This stand-alone architecture has dominated the game market for over forty years and is still the dominating architecture for many successful commercial titles today. In the middle of 1990s, networked games like

Author address: M. Zhu, A. I. Wang, H. Guo, and H. Trætteberg, Sem Sælandsvei 7-9, NO-7491, Trondheim, Norway; email: {zhumeng, alfw, guohong, hal}@idi.ntnu.no.

DOI DOI: 10.1145/XXXXXX.XXXXXX. http://doi.acm.org/10.1145/ XXXXXX.XXXXXX

Command and Conquer (C&C) and Warcraft became popular. The implementation of these games was based on the Client-Server architecture pattern, and the software was running on multiple machines connected through LAN or the Internet. The retailed game software integrated both the client and the server part of the game, so each player could either host a game on his or her machine, or join a game as a guest. The exponential growth and development of the Internet at the end of last century enabled the development of Massively Multiplayer Online Games (MMOG). Most MMOG software uses a powerful centralized server or a server farm to simulate a complex and sharable game world (virtual world), allowing multiple players to connect to the game world through the Internet with their client machines (e.g. an ordinary PC). The distribution of computation and the use of professional servers provide more computational power, which significantly improve the richness of the offered game worlds. Moreover, the staggering number of online players greatly extends the social aspects of video games and creates new opportunities for social gameplay.

Challenges have emerged during the evolution of game architectures. Work of researchers and practitioners has pointed out that poor robustness [3] and lack of flexibility [4] are among the most important challenges of the Client-Server architecture pattern. On the other hand, the Peer-to-Peer (P2P) solution, which was mainly proposed by researchers as an alternative to Client-Server, also had its drawbacks such as the difficulty in preventing cheating [3] and maintaining consistent game states on peers [5]. These challenges have motivated for further research in the game architecture field, and some sophisticated solutions have been proposed addressing these issues. However, it can be hard to capture and classify the variations between the proposed architectural solutions, as the granularity is below the level of traditional architectural patterns. Terms borrowed from general architecture field, like Client-Server, P2P, and Hybrid cannot capture the variations of the proposed game architectures. Also there are no existing game architectures or software architecture taxonomies that can be used to distinguish between the important and fine granular variations. E.g., there are several different proposed architectures that share the same name “Hybrid Architecture” [4-6]. We therefore propose a conceptual framework for game architectures with more descriptive, rhetorical, and inferential power for classifying, understanding and communicating software architectures in games on a finer granular level than what exist today. The framework can also be used to discover opportunities for future research. Although there have been published several articles on game architectures, there is very little work on what is specific about game architectures and how to distinguish between various game architectures that capture the essential characteristics of games. This article presents a framework that captures some of the core issues of game software, such as user interaction, game states, load balancing, and distribution. The idea of the framework was conceived after analyzing existing literature on game architecture and acknowledging the

difficulty in classifying the existing approaches. The main idea of our framework is to analyze how game worlds (also called game state, virtual worlds, and so forth in literature) are organized. The Game Worlds Graph (GWG) framework is based on graph theory, and two central concepts: *Game World* and *World Connector.* These concepts are the cornerstones of the framework. The framework introduces a terminology to name central concepts to provide efficient communication similar to how software developers can refer to the name of a design pattern without explaining all the underlying details. The GWG framework is first and foremost a taxonomy to classify and analyze existing architectures, but it can also be used to explore future game architectures.

The rest of this article is organized as follows: Section 2 introduces the GWG framework, Section 3 describes how the GWG framework can be used as a tool to classify existing architectures and to explore future architectures, Section 4 discusses the quality of GWG framework, the GWG Composite, and compares the framework with other taxonomies, Section 5 concludes the article.

2. THE GAME WORLD GRAPH FRAMEWORK

The initial idea of the GWG framework came from a preliminary research of exiting game architectures in order to analyze their characteristics, including their advantages and drawbacks. When we were looking into some modern game architectures, we found it difficult to distinguish between them using traditional taxonomies, e.g. typical Client-Server architecture versus Remote Rendering architecture. By examining these architectures we found that the *game state distribution and connections between the game states* are their essential differences, which inspired us to create a new architectural viewpoint: The Game Worlds Graph.

In this section, the *Game World* and the *World Connector* concepts serving the cornerstones of the GWG framework are defined. Further, the GWG formalism based on graph theory is described. Finally, the Graph Style and Flavor are defined as abstractions of commonalities among GWGs, which are further used as the basis in the game architecture classification.

2.1 Game World

We use term “World” as a metaphor to describe the game state of a video game, which is created and updated by the game program to simulate an interactive environment and its interactive objects. Typically, the *Game World* logically denotes all the dynamic objects (player characters, non-playable characters (NPCs), weapons, etc.) and static objects (buildings, rooms, sky, and so forth) in a game that are computationally represented in

system memory, and updated at run-time by the game program to reflect the newest simulation result. The Game World concept is not our invention, and has been widely used in previous research articles in terms like “World” [7], “Scenario of Game” [7], “Game State” [8, 9], “Game Gaming World” [10], “Game World” [11], and “Virtual World” [12].

Single player games and multiplayer games played on one console/PC normally have only one Game World. However, multiple Game Worlds can be identified in a game instance when looking at games with more complex architectures. E.g. a multiplayer game based on a Client-Server architecture pattern may simulate a shared remote Game World on the server, and multiple local Game Worlds on the players’ clients. Synchronization protocols are required to keep the client Game Worlds updated and consistent.

In the Client-Server example above, we find that the Game World on the server is different from those on the clients at least for two reasons: 1) The Game World on the server contains the *complete* state of the game, while the Game Worlds on the clients may only contain a local state related to a specific player; 2) When the Game Worlds are synchronized, the Game World on the server normally has higher authority that dictates the content of the Game Worlds on the clients. Neither the Game World on the server, nor the Game Worlds on the clients can be perceived directly by players. Furthermore, what players can see and hear may be different from those above-mentioned two types of Game Worlds. E.g. there may be tens of thousands of player characters simulated in the Game World on a MMOG server, only ten player characters contained in the local Game World on a specific client, and a specific player can only perceive (see and hear) five of them due to distance, view direction, and rules of the game. This third kind of Game World contains objects that are perceptible to a specific player, reflecting only externalized attributes of these game objects, in a presentable format (video stream, graphics, etc.). We have named this third kind of Game World for *Perceptible World*.

To summarize, the GWG framework differentiates between the three kinds of Game Worlds in a game with the following concepts:

1) *Global World:* The Global World (GW) contains the shared global state (all player characters, NPCs, weapons, etc.) of a game instance. The GW of a game reflects the definite, instant, and complete result of the game computation. The other Game Worlds have to be updated according to the GW sooner or later to avoid inconsistencies. There is only one *logical* GW for one running game instance. However, when it comes to the technical implementation, the GW can be implemented as multiple sub-GWs. Each sub-GW may contain partial state of the logical GW. E.g. in some MMOGs, the whole game world is divided into zones managed by different servers and each zone can be regarded as a sub-GW. Each sub-GW can also be a complete duplicate of the logical GW. E.g. in the Mirrored

Server Architecture, the same game world is simulated on multiple servers and each server manages complete and mirrored state. The duplication or partitioning of the GW is normally due to technical concerns (reduce load, decrease network latency, etc.). Since the sub-GWs contain most attributes of the logical GW, we still use the term *Global World* to denote a collection of sub-GWs.

2) *Local World:* The Local World (LW) can be regarded as the "cached" game state for one player. The LW contains only the game state that the game program considers "relevant" to a specific player , i.e. the virtual entities and the environment variables within a specific player's *area of interest*, which the player will potentially interact with between two update events. LW reflects a *dirty* game 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 games interactive, to save bandwidth, and to prevent cheating.

3) *Perceptible World:* The Perceptible World (PW) is the game world 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 games and performance concerns. Players perceive the game only through the PW externalized by various presentation devices, thus it becomes the bridge connecting the player to the logical game world.

This is not an exhaustive classification. These three concepts may not cover all possible Game Worlds, e.g. persistent Game Worlds on external memory. We choose these Game Worlds to build our conceptual base, because to our best knowledge, the essential data and processes they capture are the most relevant to the architecture inventions and innovations in the game domain.

2.2 Synchronization between Game Worlds

The Game Worlds are not isolated and the content of a Game World is usually connected with another Game World. E.g. a LW on a client must keep its content consistent with the GW on the server. We use the term *World Connector* to capture the relationship between two connected Game Worlds. The term *connector* is taken from [13], where it is defined as "a communication vehicle among components". 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 separate nodes. The communication between game worlds is considered as *update flows* through the connector. Different computations as well as different types of update flows are required for particular connector types. When two

Game Worlds are connected natively, they both will be simulated in memory on the same computing node. A connection normally involves transforming data from one format to another. A typical example is when a LW is connected to a PW. The graphical rendering computation (e.g. 3D transforming, rasterizing) is then required to draw objects onto the screen. On the other hand, if two Game Worlds are connected remotely, synchronization and other network computations are necessary. For example, when a GW is connected to a LW, the program simulating LWs has to request/receive the most recent data from the GW, and correspondingly updates LWs to provide consistent local states. Note that two Game Worlds of the same kind also can be connected. For example, in a multi-server game, each server has its own copy of the GW. State synchronization computation is needed to keep them consistent. Some solutions to this problem can be found in [5]. They are relatively more complicated than the simple synchronization protocol between a GW and a LW.

To summarize, the GWG framework defines two kinds of connectors connecting the three types of worlds (global, local and perceptible):

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

2.3 Game Worlds Graph Formalism

If we consider Game Worlds and Connectors as vertices and edges, graph theory can be used to model and relate the terms. The collection of Game Worlds and the collection of World Connectors of a game can be defined as a graph. We use the term Game Worlds Graph (GWG) to denote the graph of Game Worlds and World Connectors. The Game Worlds Graph 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.*

Two examples of Game Worlds Graphs are shown in Figure 1. 1a) is a typical GWG of a single player game, where a Perceptible World (P) is natively connected to a Global World (G), meaning that the simulation and externalization of the game world is performed on a single machine. In 1a), ***W*** = {G, P}, and ***C***= {C1}. 1b) is a typical GWG of a multiplayer online game. Three Local Worlds (L1, L2, L3) are remotely connected to a Global World (G), meaning that each player has a local copy of the global state on her or his client, which is usually a partial copy of the full state maintained on a remote server. The Perceptible Worlds (P1, P2, P3) are natively connected to the Local Worlds meaning that a client is also in charge of the externalization of the game state for a specific player. In 1b), ***W***= {G, L1, L2, L3, P1, P2, P3}, and ***C***= {C1, C2, C3, C4, C5, C6}.

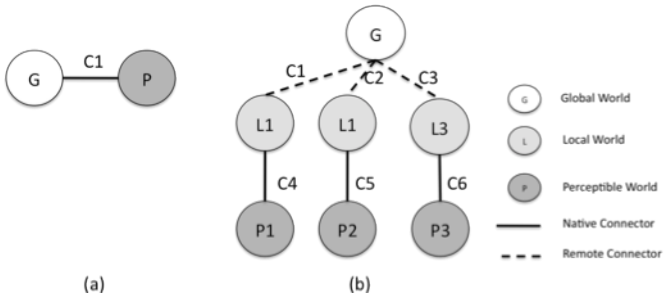

Figure 1: Two Examples of Game Worlds Graph

Game Worlds Graph serves a new perspective to how to visualize and examine the software architecture of a game. As it's defined in [13], "*The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationship among them."*

Note that the Game Worlds Graph itself will not represent the whole software architecture even if we can look on Game Worlds as elements of the game software, and World Connectors as relationships among them. However, the externally visible properties are not described. Also, "*the systems can and do comprise more than one structure and that no one structure can irrefutably claim to be the architecture"*[13]. So the Game Worlds Graph can be seen as a *view* of the game software architecture, but cannot claim to be the whole architecture. Game Worlds Graph reveals some essential architectural characteristics of a game:

1) The Game Worlds Graph provides a component-connector view [13] of the software. The Game World encapsulates an independent set of data and simulation processes typically represented by one or more components in the game software. The World Connector encapsulates the synchronization and transformation processes maintaining the connection among the components.
2) The Game Worlds Graph provides an allocation (deployment) view [13] of the software. The types of the World Connector with which the Game Worlds are connected imply how the software is assigned to computation nodes.
3) The Game Worlds Graph shows the runtime configuration of the game software, which is required for analyzing parallel and distributed performance of the game.

Note that the GWG is a runtime model of the game, and it can change from time to time. There may not be one unique graph for a particular game or even for a particular running game instance. E.g. consider a multiplayer client-server game. The number of connected players is changing continuously, meaning that the configuration of the Local as

well as Perceptible Worlds is changing, thus results in a changing Game Worlds Graph. However, the software architecture of the game does not change over time. The GWG framework provides the abstractions *Graph Styles* and *Flavors* to provide a static view that reflects the game architecture including a changing run-time configuration of players, which will be described in the next section.

2.4 Graph Styles and Flavors

A single-player game normally has no LWs (as the GW and the LW are the same), and this will not be changed during each game session. A Client-Server game may have hundreds of players connected at one moment, and the number of connected player can be thousands at another. The number of LWs is changing from time to time, however, the game always allows *multiple* LWs being created and simulated at the same time. The main difference between the GWG for a Client-Server game compared to the GWG for a single player game is whether the GWG of the game has multiple LWs or no LWs. The GWG framework provides support for describing *multiplicity* of each kind of GW. To classify GWGs according to multiplicity the following procedure is used: Once two GWGs have the same multiplicities for all three kinds of Game Worlds, we say they have the same *combination* of Game Worlds multiplicities. We use the term *Graph Style* to capture the commonality of a group of GWGs, which share the same combination. Through using 0, 1, and n to represent *zero* Game World, *one* Game World and *multiple (0 to n)* Game Worlds respectively, we can represent various configurations. According to the definition of the Graph Style, all possible Graph Styles should be represented in a list of all the possible combinations. This makes it always possible to find a Graph Style that complies with the combinations of Game Worlds and Connectors of a game by going through the list.

A Graph Style can be captured with the 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). It is a trivial task to enumerate all possible Graph Styles. However, not all possible Graph Styles make sense if we consider the common understanding of video games. E.g. the 000 Graph Style means no GW, LW and PW. Such a game does not make any sense. We have defined two hypotheses that games software must adhere to in order to represent a valid Graph Styles as described below:

H1: A video game always has a global logical state, which can be distributed to multiple software elements, but should be complete at the system level. To our best knowledge, there is no software that is publicly accepted as a game being stateless. Also game software with *only part of* the global state represented is not very practical.

H2: A video game has to present its state to player(s) in a game specific manner. Maybe a game can use an externalization mechanism other than rasterizing and

outputting on a 2D display, but it has to provide the player opportunities to perceive the game. For a pervasive game or an Alternative Reality Game (ARG), the Perceptible World may overlap with the real world, but there are still externalization mechanisms like moving an object in the real world, etc. If a software system never makes its state perceptible to the player by any means, it cannot be defined as a video game.

According to the two above hypotheses, the valid Graph Styles form the following set: {101, 10n, 111, 11n, 1n1, 1nn, n01, n0n, n11, n1n, nn1, nnn}.

The Graph Style seems to be a feasible abstraction capturing the commonalities of GWGs for creating an architecture taxonomy framework, but the information about Connectors is lost when denoting GWGs with 0, 1 or n. To address this problem, the GWG framework adds information about how the different worlds in a Graph Style are connected. The Connectors in a GWG can be zero, one or even tens of thousands, which makes it impossible to directly enumerate all Connectors in a GWG. It is therefore necessary to abstract and simplify the amount of connections and the result has to be run-time definite in order to capture the architectural characteristics. Our solution is to use the Connector types between the three kinds of Game Worlds, e.g. if all Connectors between any particular GW and any particular LW in a game are remote Connectors, we say the GWs and the LWs are *remotely* connected in the game. There are three choices of Connector types between two different kinds of inter-connected Game Worlds:

1) *Entirely remote:* All Connectors connecting any two Game Worlds of different kinds are remote Connectors (Game Worlds on separate nodes).
2) *Entirely native:* All Connectors connecting two Game Worlds of different kinds are native Connectors (Game Worlds on the same node).
3) *Hybrid:* All situations except 1) and 2).

Two game Worlds of the same kind can also be inter-connected. This means that the above classification also works for connecting Game Worlds of the same kind. The GWG framework identifies six valid combination of connection types between game worlds: 1) Connector type of GWs with GWs, 2) Connector type of LWs with LWs, 3) Connector type of PWs with PWs, 4) Connector type of GWs with LWs, 5) Connector type of LWs with PWs, and 6) Connector type of GWs with PWs.

We define the concept Graph Flavor as the mean to describe the combination of Connector type and Graph style. The Graph Flavor concept is an elaboration of the Graph Style concept based on the Connector type. E.g. one GW and one PW connected with the entirely remote Connector, and one GW and one PW connected with the entirely native Connector, can be of two different Graph Flavors, but belong to the same Graph Style (the expression is 11). The Graph Flavors of the GWG can be outlined with Expression (1).

$$\mathbf{G_m(c_g)\ c_{gl}\ L_m(c_l)\ c_{lp}\ P_m(c_p)\ c_{gp}}$$

or

$$\mathbf{G_m(c_g)\ c_{gp}\ P_m(c_p)}$$

(When there is no LW)

G_m: multiplicity of the Global World
L_m: multiplicity of the Local World
P_m: multiplicity of the Perceptible World
c_g: Connector type of the GWs with GWs
c_l: Connector type of the LWs with LWs
c_p: Connector type of the PWs with PWs
c_{gl}: Connector type of the GWs with LWs
c_{lp}: Connector type of the LWs with PWs
c_{gp}: Connector type of the GWs with PWs

(1)

When writing an expression, the following rules must be applied:

1) Use *0*, *1*, *n* to represent multiplicities;
2) Use - to represent the entirely remote Connector;
3) Use + to represent the hybrid Connector;
4) For entirely native Connector, if it is
   i. **c_{gl}, c_{gp}** or **c_{lp}**, ignore the Connector type.
   ii. **c_g**, **c_l**, **c_p**, use = to represent the Connector type.

Table 1 shows some examples of Graph Flavor expressions.

Table 1 GWG Game Flavor Expression Examples

Expression Examples	Meaning
1-nn	One GW is remotely connected with multiple LWs, and the latter are natively connected with multiple PWs. The expression represents the GWG Flavor of a typical Client-Server game.
11	One GW is natively connected with one PW. The expression represents the GWG Flavor of a typical single player on single machine game.
n(-)-nn	Multiple GWs are inter-connected with each other remotely, and are remotely connected with multiple LWs. The latter are natively connected with multiple PWs. The expression represents the GWG Flavor of a typical multi-server, multiplayer, Client-Server game.

The definition of the GWG forms the foundation layer of the GWG framework, while the Graph Style and the Graph Flavor concepts form the higher abstraction layers.

Based on these two concepts, we have created a taxonomy framework for classifying game architectures, which provides a new view of game architectures. The view is related to traditional architectural patterns, for example the Model-View-Controller (MVC) pattern. The GWG framework can be mapped to the MVC pattern as the GW and the LWs represent the *model*, the PW is the *view*, and the process supporting connectors represents the *controller*. However, the GWG is still different from MVC because it represents details on a finer granularity level where the state distribution and connections play the central roles. These issues are not covered by the MVC pattern since MVC does not (also is not intended to) differentiate between the various models, views, and controllers. The lower level of abstraction makes the GWG a taxonomy that is able to represent game architecture at a more refined level than traditional pattern based taxonomies. Moreover, the Graph Style and Flavor are useful for exploring unknown and unused game architectures, which is demonstrated in Section 3.4.

3. USE OF THE GWG FRAMEWORK

This section describes the use of the GWG as a taxonomy framework based on the Graph Style and the Flavor concepts. To show the validation and usefulness, we outline a systematic review of game architectures, and present three examples. Another application of the GWG is a tool to explore future architectures, this is also discussed in the section and further illustrated with an example.

3.1 GWG as an Architectural Taxonomy Framework for Video Games

Software architectures for games vary and current taxonomies are too rough to capture the specific characteristics that make the game architectures novel. As we have discussed in Section 2.3, the GWG reflects on some essential aspects of game software architectures, and therefore it can provide some new perspectives.

Assuming that we can always identify the GWG, the Graph Style and the Flavor of every game architecture, then the architectures with the same Graph Style naturally form an architectural category. Further, a classification of game architectures can be done through identifying such categories. 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 through the same process. These two classifications form two layers of the GWG taxonomy framework providing two levels of granularity. Figure 2 shows the framework hierarchy. In the Graph Style Layer, the Graph Style expressions like 101 and 10n are used to name the architecture kinds. While in the Flavor layer, the Flavor expressions like 1-1 and 1-nn are used to name the architecture categories.

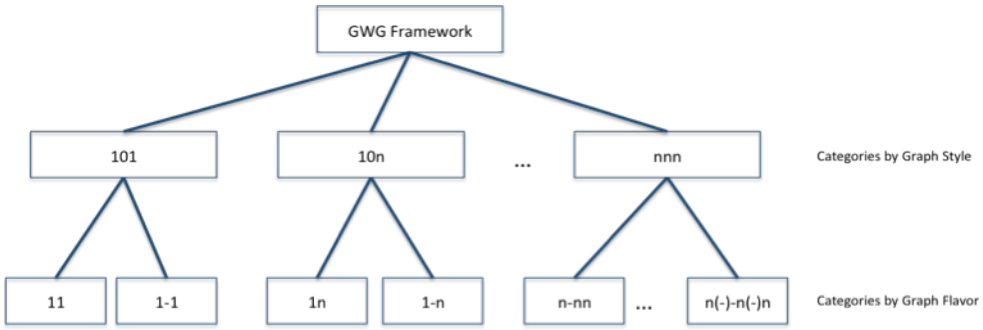

Figure 2: GWG as an Architectural Taxonomy Framework

It is difficult to theoretically prove our assumption that GWGs can always be identified in various described architectures. It is also difficult to systematically validate the framework, since it requires a thorough review of existing game architectures, which is beyond the scope of this article. However, to empirically show the validity of the framework, we will analyze the architectures of three games described in the literature based on the Graph Style and the Flavor (see Section 3.2). Before taking a closer look at these examples, we first describe the process of classifying architectures, which includes following steps:

- Step 1: Identify the three kinds of Game Worlds, and their Connectors in the game architecture, and then draw a GWG of the architecture.
- Step 2: Generalize the GWG by identifying multiplicities of the three kinds of Game Worlds, and combine the multiplicities to present the Graph Style of the architecture. The architecture can thus be fitted into the corresponding category in the upper layer of the GWG taxonomy framework.
- Step 3: Generalize the Connector types between the Game Worlds by summarizing possible Connectors between combinations of the Game Worlds. Combine the information of the Connector types with the Graph Style, and the elaborated Graph Style represents the Graph Flavor of the architecture. The architecture can thus be fitted into the corresponding category in the lowest layer of the GWG taxonomy framework.

The process will be illustrated through three examples in the following section.

3.2 A Systematic Review Based on the GWG Framework

To prove the capability of the GWG Framework as a general architectural taxonomy for video games, solid evidence was required. Our approach was to carry out a systematic review of game architectures in literature. We searched four electronic databases and browsed through 2766 hits. By filtering the hits with inclusion and exclusion criteria, we included 40 articles in the final review. The architectures described in the articles were

analyzed from the GWG perspective, and then fitted into the GWG taxonomy framework. The architectures ranged from single player games to multiplayer networked games, and the architectures are representative for the field to our best knowledge. Our hypothesis was that the architecture of the games found in the research literature would only fit into one of the Graph Style categories, and into only one of the Flavor categories.

The systematic review method we used was adapted from [14], and it was tailored according to our requirements. Due to the limitation of the length of this paper, we only provide some of the relevant results in this article. The complete review is documented in another paper [15]. The four electronic databases we searched were ACM Digital Library, Compendex, IEEE Xplore, and ScienceDirect –Elsevier, and we used the keywords: **(Game** *OR* **Gaming)** *AND* **Architecture**. The keywords were searched for in the title, the abstract and the keywords of sources published between 1990 and 2009. Our searches ended up with 2766 "hits". All of these hits had to be checked, and irrelevant papers were excluded. We defined relevance criteria to identify candidate articles, and after three steps of filtering, 40 papers that all described a game architecture were included. The classification results are presented in Figure 3, where all the possible GWG Graph Styles are enumerated and the number of architectures found for each Graph Style is reported.

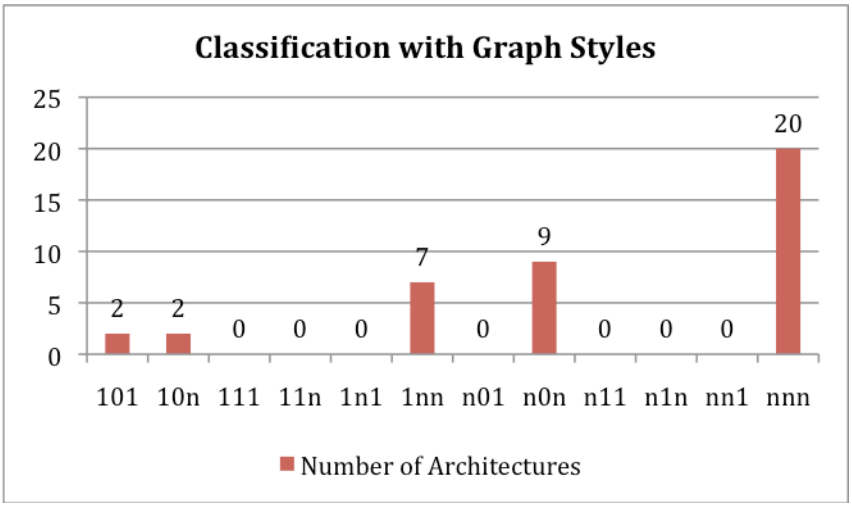

Figure 3. Classification with Graph Styles

From the results we see that the game architectures found in research literature are centered on the Graph Styles 1nn, n0n and nnn, and the latter gets the most attention. We also found a few papers that described game architectures that fall into the 101 and 10n Graph Styles. Another observation was that 7 categories of Graph Styles are not present in research articles, showing that the research on game architectures focus only on a limited set of Graph Styles. We believe the following reasons can explain the focus on few Graph Styles:

1) Research on game architectures focus on proposing innovative software architectures and not relatively mature and standard architectures such as architectures for stand-alone games (no network support).

2) Not all of the Graph Styles are practical. For example n1n Graph Style implies multiple Perceptible Worlds with one Local World, which describes a single player-multi-view game. However, it is impractical to have multiple Global Worlds to support such a game style.

3) The Graph Style captures important attributes of game architectures, but the classification is still at a high conceptual level. This means that there can be several differences between software architectures within the same Graph Style. GWG Flavor can be used to capture some of these differences on a lower abstraction level.

We also analyzed available research literature for GWG Flavors. If a GWG Flavor was present in an article; a Flavor sub-category in the corresponding GWG Graph Style category was created. The GWG Flavors classification results are presented in Figure 4.

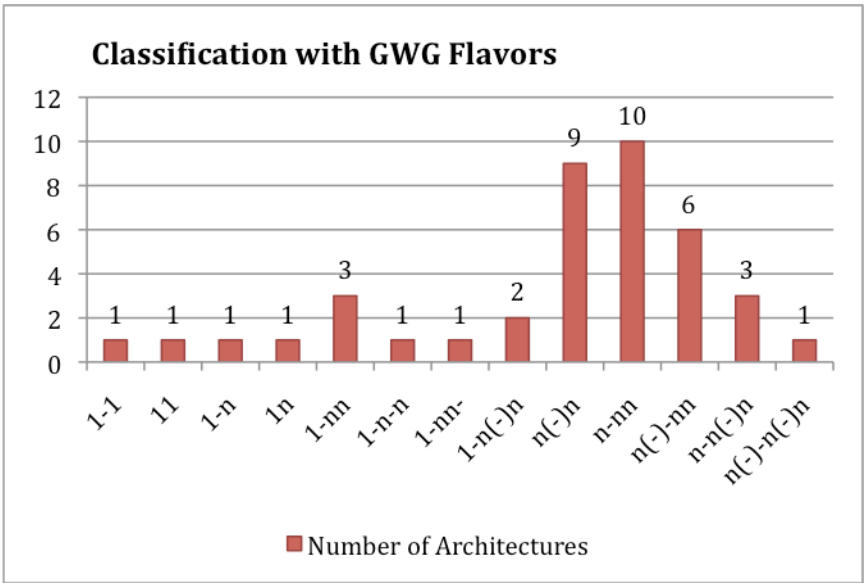

Figure 4 Classification with GWG Flavors

Figure 4 shows that the 40 architectures are fairly evenly distributed among the GWG Flavor categories, with a higher concentration around the categories n(-)n, n-nn and n(-)-nn. These three architectural styles characterize two kinds of Multi-Server architectures and a Peer-to-Peer architecture. This observation adheres to our knowledge about the focus of game architecture research. There are still more possible Graph Flavors that are not present in the reviewed articles, and some of them reveal potential for new architecture innovation. This will be discussed in Section 3.4, and Section 4 describes a validation of the inferential power of the GWG framework. The complete list of the reviewed sources are

documented in the bibliography of [15], and the architectures for each category/sub-category are also presented with a table in the same paper.

To further show the usefulness of the GWG framework, the next section describes and analyses of three game architectures selected from Figure 3. The selected examples are picked from the three Graph Style categories that are most commonly described in the literature. Thus, these examples reflect the use of GWG on the architecture styles game researchers are most interested in.

3.3 Using GWG as a Taxonomy Framework for Analyzing Three Game Architectures

The examples of game architectures used in our analysis were selected from the top three most popular categories identified in the review. The categories are 1nn, n0n, and nnn. The architectures represented in these three categories are Client-Server, Peer-to-Peer, and Multi-server architectures respectively.

Game architecture 1: Strifeshadow Fantasy (SSF) [16] is 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 application. The HTTP protocol is used for client-server communication, and Macromedia Flash 4.0 is used to render the game scene and regularly sends Update Request Message (URM) to the server. The server is based on Active Server Page (ASP) combined with HTML, scripts, and Microsoft ActiveX server components. The server uses a relational database management system to store and retrieve the game state. In SSF, a unique Global World is maintained on the server while each player connected to the server has a client managing a Local World on his/her device. The Perceptible Worlds are also presented on players' devices. Figure 5 shows the GWG of the SSF game.

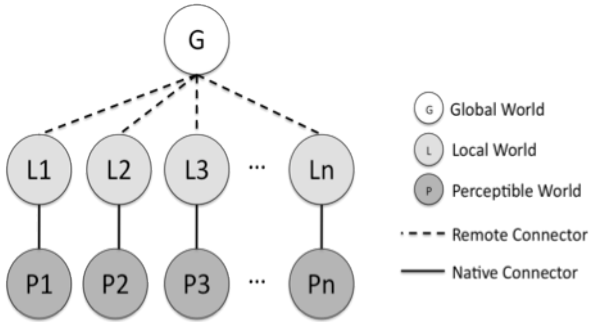

Figure 5: The GWG of Strifeshadow Fantasy

The multiplicity of the Global World of SSF is "1", while the Local and the Perceptible Worlds are "n". This means that SSF falls into the Graph Style 1nn category in

the GWG Framework. The Connector type between the Global and the Local Worlds is entirely remote while the Connector type between the Local and the Perceptible Worlds is entirely native. This means that the SFF falls into the 1-nn Flavor category.

Game architecture 2: VoroGame [17] uses a structured P2P overlay based on a distributed hash table (DHT) (e.g.[18, 19]) combined with an unstructured P2P overlay based on a Voronoi diagram [20] to manage the global state. The Game World is divided into zones based on 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 simulating the behavior of objects inside the zone that the player of the peer locates in, and 2) *DHT responsible* that is in charge of storing objects and forwarding objects state changes from Voronoi responsible to players that require information about the objects state. Figure 6 shows an example of the GWG of the VoroGame. Note that the number of players changes dynamically, so there can be various GWGs for this architecture. However, the game will be categorized in only one Graph Style and one GWG Flavor.

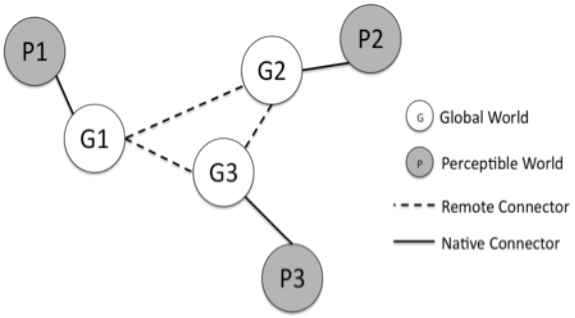

Figure 6: An Example GWG of VoroGame

The multiplicity of the Global and the Perceptible Worlds of the VoroGame are “n”. The Local World is not presented in the architecture. So the Graph Style of GWG of the VoroGame is n0n. The Connector type between the Global and the Perceptible Worlds is entirely native, and the Connector type between the Global Worlds is entirely remote. This means that the VoroGame falls into the n(-)n Flavor category.

Game architecture 3: Rokkatan [21] is a multiplayer online game based on a Mirrored Server architecture. The game state is simulated on multiple servers, and every server manages a full instance of the entire game state. The servers use a cooperation protocol to keep the game state consistent. Players joining the game can select the fastest server to get a relatively smooth gaming experience. Players interact with the Global World through their client machines where Local Worlds are simulated and the game is presented to the user (Perceptible World). Figure 7 shows an example the GWG of Rokkatan. Note that

there may be various ways of organizing servers and thus corresponding variations of Global Worlds structures.

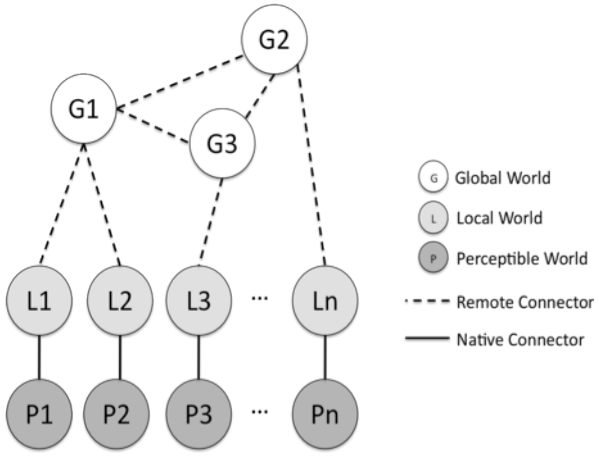

Figure 7: a GWG of Rokkatan

The multiplicity of the Global Worlds of Rokkatan is "n", and that of the Local Worlds and the Perceptible Worlds are also "n". This means that the Rokkatan game falls into the nnn Graph Style category. The Connector type between the Global Worlds is entirely remote. The Connector type between the Global and the Local Worlds is also entirely remote, while the Connector type between the Local and the Perceptible Worlds is entirely native. This means that the Rokkatan game falls into the n(-)-nn Flavor category.

3.4 Use the GWG framework to Explore Future Architectures

Besides classifying existing game architectures, the GWG framework can also help in exploring future architectures. A general process of this application is described below:

- Step 1: Examine the distribution of existing architectures in the GWG categories (Graph Style based) and sub-categories (Flavor based), and identify the unfilled categories/sub-categories.
- Step 2: Each unfilled category/sub-category describes a potential high-level *architecture pattern* (only architectural characteristics related to the GWG are described). By examining the Graph Style/Flavor of the category/sub-category, we can reason about the major features of the architecture pattern.
- Step 3: Matching the features of the architecture pattern to the architectural drivers [13] connected to each game type to justify if these characteristics are appropriate for a specific game type.
- Step 4: If an architecture pattern is considered appropriate to a game type, add more architectural tactics that are relevant to the game type to make it a concrete architecture.

- Step 5: Evaluate the architecture with either empirical or theoretical methods.

The steps 1-2 are directly supported by the GWG framework itself, while the steps 3-5 require cooperation with other architectural methods. This means that the role of the GWG framework in this innovative process is to be a tool for generating the new high-level architecture patterns. We use an example to illustrate how the GWG can be applied in such a way in the rest of the section.

Through analyzing Figure 3 and Figure 4, we find that in the Graph Style 1nn category, there are four GWG Flavors identified in existing architectures, which are 1-nn, 1-n(-)n, 1-n-n and 1-nn-. It is a trivial task to enumerate the possible GWG Flavors in the Graph Style that are not featured by any existing architecture, for instance, 1-n+n, 1-n(-)-n, and etc. We will further explore 1-n+n as an illustration.

The 1-n+n Flavor implies that multiple players interact with a single, shared Game World (one GW) through their individual views (multiple PWs), and the Remote Connector type between the GW and the LWs reveals that the GW is simulated on an individual node. Since the PWs are always presented on the machines players directly interact with, the hybrid Connector between the LWs and the PWs implies two kinds of player machines, one doing local state simulation, rendering and presentation, and the other (thin clients) only doing presentation. The thin clients connect to the rendering servers through a network, and the rendering servers maintain the local game state (LWs) and stream the rendering results to the thin clients. The characteristics of the architecture pattern implied by the 1-n+n Flavor can be summarized as follows:

1) Single centralized machine simulates the global state.
2) Local state is maintained on multiple machines for rendering and player interaction.
3) The above two kinds of machines are connected through a network.
4) Players interact with two kinds of machines, one is a fully-functioning client machine in charge of local state simulation, rendering and presentation, and the other is a thin client machine only in charge of presentation.
5) Remote rendering is done on rendering servers, which maintain their own local game state. The game server is dedicated to the global state simulation.

The characteristics 1- 3 are the same as traditional centralized server architectures, but the characteristics 4 and 5 make the architecture pattern novel. These characteristics describe that local and remote rendering are supported at the same time, meaning that client devices with different computational capabilities can be used. The remote rendering is done on rendering servers, not on the GW server, which differentiates the architecture pattern from its 1-nn- siblings. The architecture is appropriate for mobile ubiquitous games, which run on heterogeneous devices. Further, if we use some of the clients that are capable of rendering

as the rendering servers for the thin clients, the architecture will have the advantages of Public Server architectures, thus can provide better scalability and save investments on rendering servers.

As a summary, the architecture pattern can be elaborated by defining three kinds of client devices: i) Fully-Functioning Client (FFC), ii) Thin Client (TC) and iii) Super Client (SC). Figure 8 shows the architecture overview.

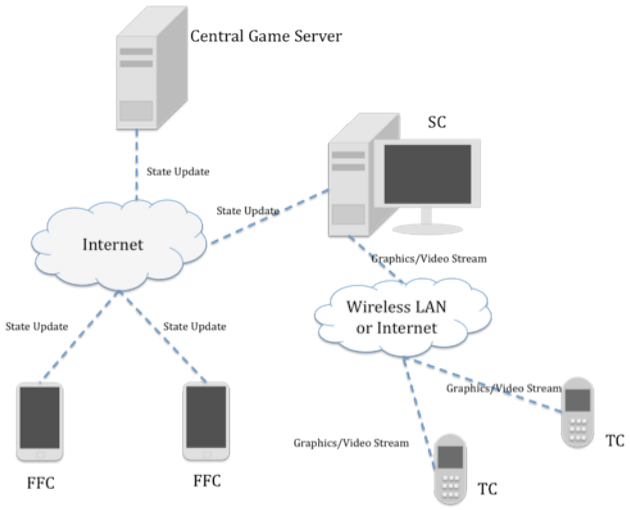

Figure 8. A Hybrid Remote-Rendering Architecture with 1-n+n GWG Flavor

In Figure 8 we see that high performance smart phones are used as the FFCs, desktop PCs are used as SCs and low-end cellphones are used as TCs. Each TC is connected to a SC, which generates visual game scene as graphics/video stream for it. The central game server delivers state updates to FFCs and SCs. The FFCs having full graphics capabilities, maintain their local state and render the game world. The SCs are more powerful, thus they maintain not only the game state for their direct users but also the game state for the connected TC users. The SCs render the Game World for their users and generate graphics/video streams for TC users in parallel. The TCs do not directly process the state update from the central game server, and they connect to SCs to fetch the streamed scene for playback. For players' inputs, FFCs and SCs both directly handle player input and communicate with the central game server, while TCs only forward the input to the connected SCs since the input handling needs the local state as context. To make it simple, the data stream from the clients to the server is not presented in Figure 8.

4. DISCUSSION

Previous section presented how the GWG framework can be applied to existing game architectures and how our taxonomy can distinguish between different types of game architectures at a finer granularity level than other existing taxonomies. This section will

discuss and analyze the quality of the GWG framework. The term *quality* means the appropriateness and the usefulness of using the GWG framework for understanding game architectures from a pragmatic perspective. The *quality* evaluation is analysis-based by adapting a framework for evaluating theories from the Computer Supported Collaborative Work (CSCW) domain. In addition to analyzing the *quality of the framework,* practical experiences and issues from using the GWG to analyze architectures of games in general will be shared.

4.1 The Quality of the GWG Framework

In [22], Halverson proposes a framework for evaluating a theory from pragmatic view. The framework is used to evaluate theories in the CSCW domain. However, we argue that such an evaluation can also be adopted to evaluate the concepts and foundation of our GWG framework. In Halverson's framework, four attributes of theory are identified which are required by the people who use it. The attributes are: 1) *Descriptive* power – Theory should provide a conceptual framework that helps us make sense of and describe the world; 2) *Rhetorical* power – Theory should help us talk about the world by naming important aspects of the conceptual structure and how it maps to the real world; 3) *Inferential* power – Theory should help us make inferences; and 4) *Application* - How we can apply the theory to the real world for essentially pragmatic reasons.

In the GWG framework, Game World and the World Connector are defined as the basic concepts, building the formalized foundation for the concept of Game Worlds Graphs. The GWG reflects some essential characteristics on how game software is organized. The GWG can be regarded as a means to *describe* the game architecture domain. Moreover, the Graph Style and the Graph Flavor are proposed as abstractions of the basic GWG concepts, which *describe* the game architecture at a higher abstraction level. The Graph Style and the Graph Flavor serve as the cornerstones of the taxonomy framework, which *describe* the commonalities and the differences of game architectures. The framework is sufficiently *descriptive* to not only describe variations of the architectures of networked games as most of the taxonomies can do (but at a higher abstraction level), but it also categorizes the architectures of single player games. The validity of the GWG as a taxonomy framework is supported through a systematic review where some of the relevant results are provided in Section 3.2. Further, Section 3.3 shows how the GWG framework can be used to analyze three different game architectures and how our framework highlights their differences. The GWG framework can describe the game architectures as well as the classification of the game architectures. The above arguments demonstrate the *descriptive* power of the GWG framework.

The GWG framework provides a terminology for naming the Graph Style and the Flavor. The terminology naturally works for naming the categories in the GWG taxonomy framework, since the categories are based on the Graph Style and the Flavor. The terminology includes two layers:

1) Graph Style layer, where expressions like 101, 10n are used to denote the architectures with information on Game Worlds distribution.
2) Flavor layer, where more complicated expressions like 1-1, 1-nn are used to denote the architectures with information on both Game Worlds distribution and Connector types.

The expressions are informative and efficient since essential characteristics of game architectures (Game Worlds distribution and Connector types) are presented through simple expressions. The expressions are intuitively more concise and expressive than existing terminology (e.g. Client-Server, Peer to Peer and Hybrid), as they reveal important architectural differences that architectural pattern names currently do not capture. The expressions are layered so they can provide two levels of granularities allowing them to be more adaptive to wider domains. The capability of the terminology in naming phenomena in game architecture shows the *rhetorical* power of the GWG theory.

The GWG framework can help reason about quality attributes of architectures, when they are fitted into the framework. E.g. for n(-)-nn architectures, the duplication of the Global World can usually avoid single point of failure, which results in better availability than 1-nn architectures. Sometime the inferences can even be about the characteristics of *unknown* architectures. In Section 3.4 we presented an example of using the GWG framework to explore a Hybrid Remote-Rendering architecture, which to our best knowledge has not been proposed in any previous research article. Although the described architecture still needs to be refined to solve issues such as incentive mechanism and cheat prevention, the exploration has provided an architecture design prototype. More proposals of new game architectures can be made using the GWG framework. E.g. in the n0n category, we can reason with the GWG framework that possible flavors may include n-n, n(-)-n, n(-)n, and so forth. However, only n(-)n architectures are presented in articles we reviewed. With the analysis based on the GWG, we foresee that the n(-)-n architecture can be a derivation of the Mirrored Server Architecture, where the Remote Rendering techniques are used to remove rendering computation from the client. We believe such *explorations* will result in proposals of new architecture patterns that can be useful and reveal new opportunities. A general approach of using the GWG to explore unknown architectures is to enumerate possible Graph Styles and Flavors, and identifying categories that make sense but that have not yet been implemented in games. Thus, new architectures based on identified GWG Graph Styles

and Flavors can be defined and explored. The GWG framework opens for exploration into new territories of existing and future architectures and which proves the *inferential* power of the GWG framework.

When it comes to *Application*, at least three opportunities can be identified:

1) *Classifying exiting architectures:* GWG framework includes a taxonomy which defines categories of game architectures. By fitting existing architectures into this framework, the GWG theory can improve the understanding of existing architectures based on their common and unique characteristics. The results of this application are described in [15].
2) *Sense-making and communication:* GWG framework defines a terminology to communicate and make sense of each category in the GWG framework, which is similar to how software architects can refer to design pattern without explaining all the details. E.g., using the terms 1-1 and 1-nn can concisely and precisely indicate the commonalities of architectures in a category. The research on game architectures is still an immature domain with few frameworks, defined processes, methods and patterns. The GWG framework contributes to provide a new architectural perspective on game architectures as well as an efficient language for discussing architectural differences and similarities.
3) *Exploring unknown architectures:* By enumerating possible Graph Styles and Flavors, researchers can explore non-existing architectures using the GWG framework. The example in Section 3.4 showed how this is achieved with the GWG.

4.2 GWG Composite

Games today often provide different modes in how players play the game that includes both single player, multi-player on same machine and multi-player-online, which are implemented using different architecture patterns. An example is Ubisoft's action game Splinter Cell: Double Agent, which has a single player mode and a multiplayer mode. In the single player mode, player unfolds a story by completing missions, and the architecture is the typical stand-alone architecture, which falls into the 11 category. In the multiplayer mode, players use client machines with network connection and compete against each other by achieving predefined goals, and the architecture is a Client-Server architecture, which falls into the 1-nn flavor. The example reveals that sometimes it is not possible to describe the architecture of a game only with *one* architectural pattern. In other words, some games can hardly be regarded as *one* game from the architectural point of view as they feature totally different game modes. We define the term *composited architecture* to denote this phenomenon. The term *GWG composite* captures the characteristic of the GWGs for

composited architectures in one game. A GWG composite can consist of two or more GWG Graph Styles and Flavors, e.g. 11 + 1nn and 11 + n-nn.

Understanding of GWG composite can help us investigate the game architecture from a higher level of abstraction. When we technically study the architecture of a game, we tend to simplify design constraints and user demands on gameplay, thus we generalize the architectures with abstract patterns like Client-Server, Peer to Peer, etc. However, when developing a game, the composite architectures can usually help in building practical insight. The GWG composite is capable of describing the composited architecture, and the GWG composite can also help analyzing the appropriateness of different composited architectures before the design decision is made. Further analysis of GWG composite of several existing games can also be used to identify if there are technical challenges in integrating various GWG Styles and Flavors in one game, and possible solutions for dealing with such integration. This is another unexplored area of research initiated by the work on the GWG framework.

4.3 COMPARISON WITH EXISTING TAXONOMIES: THE RELATED WORK

Most of the taxonomies in literature focus on networked games and the categories are defined in an ad-hoc way. In [5], architecture considerations on MOGs were discussed, where game architectures were summarized as i) Client-Server, ii) Peer-to-Peer, and iii) Hybrid architectures. The author used the term "Hybrid" to describe the Mirrored Server Architecture. This taxonomy does not cover games other than networked games, and it uses the "Hybrid" category, which is an ambiguous term, to cover any variant that does not fall into the Client-Server or Peer-to-Peer categories. For instance, different architectures were named "Hybrid" in [4-6]. Similarly, in [23], the game architectures were classified into three categories: i) Client-Server, ii) Peer-to-Peer, and iii) Federated Peer-to-Peer. The latter is a Peer-to-Peer architecture with group management, where players in the same interaction group form a multicast group, and Application Level Multicast (ALM) is used for communicating within the group. This taxonomy does not cover non-networked games, and does not cover Multi-Server architectures.

Another taxonomy was introduced in [3], where four categories were defined: i) Peer-to-Peer, ii) Client-Server, iii) Mirrored-Server, and iv) Clustered-Server. In this taxonomy, multi-server zone-based architectures and mirrored server architectures were distinguished. A more elaborated taxonomy was proposed in [24], with five categories defined: i) Client-Server, ii) Peer-to-Peer, iii) Client-Multi-Server, iv) Peer-to-Peer with Central Arbiter, and v) Mirrored-Arbiter. The GWG is different from these taxonomies at least for three reasons:

1) The GWG is a taxonomy framework intending to cover *all* game architectures, which is not restricted to only networked games.
2) The GWG is a systematic framework with inferential power.
3) The GWG provides a more informative, unambiguous and efficient terminology.

The taxonomy proposed in [25] is a framework that is the one closest to the GWG. The framework also uses the graph concept and defines the term *communication graph*. However, for communication graphs, the node is the computing node and the edge is the network communication. This framework is different from the GWG framework in the following points:

1) The node in the GWG framework is defined with the Game World, while the node in the communication graph is defined with the computing node. So the GWG framework focuses on the state distribution, and the communication graph focuses on the computation distribution.
2) The edge in the GWG framework describes the synchronization between the Game Worlds, while the edge in the communication graph describes the network communication. The synchronization is more general than network communication, which can be the network communication but is not restricted to network communication.
3) The GWG identifies Perceptible World and is able to describe how the user interface is related to the rest of the game software. It can therefore describe a wider variety of architectures such as remote rendering architectures.
4) The GWG framework defines Graph Style and Flavor as the abstraction of commonalities of GWGs. These concepts can help infer attributes of existing architectures and build insight into future architectures.
5) The GWG framework provides an informative terminology to name categories.

In [26], a taxonomy is introduced to characterize pervasive games/applications. This taxonomy focuses on abstracting application characteristics, independent of the characteristics of the middleware or infrastructure that support the application, and provides a controlled vocabulary for thinking about the application. These characteristics may impact the application architecture. The taxonomy is not directly about architectures, but it can be used in framing discussions about potential designs and architectures of applications. We think the potential synergy of using the taxonomy with GWG in exploring future architectures is worth further studies.

5. CONCLUSION AND FUTURE WORK

The major contribution of our work is the invention of the Game Worlds Graph framework. The GWG is a conceptual framework based on the Game World and the Connector concepts. The framework provides a new perspective on game architectures and reveals some essential characteristics of game architectures. Thus, the framework is useful for getting a deeper understanding of game architectures. The Graph Style and Flavor are two concepts to capture commonalities of GWGs, and a taxonomy framework for game architectures is defined based on these concepts. To name the categories in the taxonomy, a terminology is proposed, which is more informative, descriptive and efficient than existing ones. Our framework provides new terminology for discussing and categorizing software architectures without the need of explaining all the details of the architectural patterns used. The validity of the taxonomy framework is backed up by being able to distinctly classifying the majority the described game architectures in the literature from 1990 to 2009 more elaborate than done before, and as well showing the validity through a detailed analysis of three game architectures.

Apart from being a tool for analyzing existing architectures, the GWG framework can be used to explore future architectures. The Graph Styles and Flavors that are not identified in known architectures provide hints on potential architectural patterns, and can thus help create and analyze novel architectures for modern games and even future games. For instance, the GWG framework is capable of analyzing various cloud-based architecture for games by considering various configurations for how the game state is distributed in the cloud and on the clients, as well as how the presentation of the game is handled (streaming vs. client presentation). The quality of the GWG was further evaluated based on an evaluation framework adapted from the CSCW domain. The evaluation argues for the descriptive power, the rhetorical power, the inferential power and the application of the GWG framework. Furthermore, the GWG composite concept is included to address the problem when a game consists of several game modes with different architectural characteristics.

The computer game evolves with the change of technical, social and cultural context, and the GWG framework also needs to be updated continuously. The interesting improvements of the framework we can foresee include identifying more kinds of Game Worlds and Connectors, and adding concepts that can make the framework more powerful. Future work includes developing and evaluating concrete new architectures that are revealed by the GWG. For improving the framework, the update flows through connectors should be further researched. We are also thinking about connecting Game Worlds with software models to bridge the GWG with modeling approaches to make game development faster and easier.

ABOUT THE AUTHORS

Meng Zhu is now a Ph. D. Student in Department of Computer and Information Science, Norwegian University of Science and Technology (NTNU), Trondheim, Norway. Before starting his Ph. D., he worked in Ubisoft as a game developer, and developed famous game titles including Splinter Cell: Double Agent and Rayman Ravin Rabbits for the XBOX 360 in Ubisoft's leading development teams. He is doing research on Mobile and Social Games. His research interests include Pervasive Games Concepts and Techniques, Software Architecture, and Model Driven Development.

Alf Inge Wang is a professor at the Norwegian University of Science and Technology (NTNU). He received his Master of Software Engineering in 1995 from the NTNU and his PhD from the same university in 2001. Wang has over 70 peer-reviewed international publications and is head of NTNU's research program on video game, head of JoinGame - a professional network of game developers and game researchers, and is the editor of the event calendar of ACM's Computers in Entertainment. His research interests include software engineering education, game technology, social computing, mobile computing, and software architecture.

Hong Guo is a Ph. D. student in the Norwegian University of Science and Technology. Her current research focuses on the concepts and modeling techniques for pervasive and social games.

Hallvard Trætteberg is an associate professor at the Department of Computer and Information Science at NTNU and a member of the Information Systems group. His research interests are model-based user interface design and model-driven development of information systems including games.

ACKNOWLEDGEMENTS

The first author wants to thank his colleague Bian Wu for his input in the brain storming which motivated the article.

REFERENCES

1. DeMaria, R. and J.L. Wilson, *High score! The Illustrated History of Electronic Games,* . 2 ed2004: McGraw Hill.

2. Blow, J., *Game Development: Harder Than You Think.* Queue, 2004. **1**(10): p. 28-37.
3. Hsu, C.-C., et al. *On the Design of Multiplayer Online Video Game Systems*. 2003. Orlando, FL, United states: SPIE.
4. Huey-Ing, L. and L. Yun-Ting. *DaCAP - A Distributed Anti-Cheating Peer to Peer Architecture for Massive Multiplayer On-line Role Playing Game*. in *Cluster Computing and the Grid, 2008. CCGRID '08. 8th IEEE International Symposium on*. 2008.
5. Roccetti, M., S. Ferretti, and C.E. Palazzi. *The Brave New World of Multiplayer Online Games: Synchronization Issues with Smart Solutions*. in *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*. 2008.
6. Cecin, F., et al. *FreeMMG: a hybrid peer-to-peer and client-server model for massively multiplayer games*. 2004. ACM New York, NY, USA.
7. Bouras, C., et al., *Networking Aspects for Gaming Systems*, in *The Third International Conference on Internet and Web Applications and Services*2008.
8. Wierzbicki, A. and T. Kucharski. *"P2P scrabble. Can P2P games commence?"*. in *Peer-to-Peer Computing, 2004. Proceedings. Proceedings. Fourth International Conference on*. 2004.
9. Mauve, M., S. Fischer, and J. Widmer, *A generic proxy system for networked computer games*, in *Proceedings of the 1st workshop on Network and system support for games*2002, ACM: Braunschweig, Germany.
10. Hampel, T., 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*2006, ACM: Singapore.
11. Kabus, P., et al., *Addressing cheating in distributed MMOGs*, in *Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*2005, ACM: Hawthorne, NY.
12. Rhalibi, A.E. and M. Merabti, *Agents-based modeling for a peer-to-peer MMOG architecture.* Comput. Entertain., 2005. **3**(2): p. 3-3.
13. Clements, P. and R. Kazman, *Software Architecture in Practices*2003: Addison-Wesley Longman Publishing Co., Inc. 528.
14. Dybå T and T. Dingsøyr, *Empirical studies of agile software development: A systematic review.* Information and Software Technology, 2008. **50**(9-10): p. 833-859.
15. Meng Z., W. Alf Inge, and G. Hong, *From 101 to nnn: A Review of Computer Game Architectures.* Will be submitted to ACM Computers in Entertainment.
16. Chan, H.T. and R.K.C. Chang. *Strifeshadow Fantasy: a massive multi-player online game*. in *Consumer Communications and Networking Conference, 2004. CCNC 2004. First IEEE*. 2004.
17. Buyukkaya, E., M. Abdallah, and R. Cavagna. *VoroGame: A Hybrid P2P Architecture for Massively Multiplayer Games*. in *Consumer Communications and Networking Conference, 2009. CCNC 2009. 6th IEEE*. 2009.
18. Rowstron, A. and P. Druschel, *Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems*, in *Middleware 2001*, R. Guerraoui, Editor 2001, Springer Berlin / Heidelberg. p. 329-350.
19. Stoica, I., et al., *Chord: A scalable peer-to-peer lookup service for internet applications.* SIGCOMM Comput. Commun. Rev., 2001. **31**(4): p. 149-160.
20. Aurenhammer, F., *Voronoi diagrams- a survey of a fundamental geometric data structure.* ACM Comput. Surv., 1991. **23**(3): p. 345-405.
21. Muller, J., et al., *Rokkatan: scaling an RTS game design to the massively multiplayer realm.* Comput. Entertain., 2006. **4**(3): p. 11.
22. Halverson, C.A., *Activity Theory and Distributed Cognition: Or What Does CSCW Need to DO with Theories?* Computer Supported Cooperative Work (CSCW), 2002. **11**(1): p. 243-267.

23. Bauer, D., et al. *Communication architectures for massive multi-player games*. 2004. Kluwer Academic Publishers.
24. Yang, L. and P. Sutinrerk, *Mirrored arbiter architecture: a network architecture for large scale multiplayer games*, in *Proceedings of the 2007 summer computer simulation conference*2007, Society for Computer Simulation International: San Diego, California.
25. Smed, J., T. Kaukoranta, and H. Hakonen, *Aspects of networking in multiplayer computer games.* Electronic Library, 2002. **20**(2): p. 87-97.
26. Dombroviak, K.M. and Ramnath, R., 2007, *Taxonomy of Mobile and Pervasive Applications*. *Proceedings of ACM Symposium on Applied Computing*, pp. 1609-1615