Automated networking in Meta-Casanova

1 Introduction

Adding multiplayer support to games is a desirable feature to increase the game popularity by attracting millions of players [2]. The benefit for the popularity is given by the fact that, after completing the singleplayer experience, the user may keep playing the game online in cooperation or against other players, in different modes that are more interesting and challenging than their singleplayer counterparts [5]. Multiplayer games are very sensitive to network delay (latency), packet loss, and bandwidth limits, thus requiring heavy optimizations that are different and depend on the specific instance of the game. For instance, a Turn-based strategy game will require less optimization with respect to bandwidth usage or delays, because the data transmission happens once.

To be continued...

2 Challenges of multiplayer games

In this section we explore several existing solutions to designing a networking layer for multiplayer games. For each solution we list the main advantages and drawbacks, and finally we present our approach and formulate a problem statement.

2.1 Preliminaries

Architecture of networking in games, local view of the "real" game state, centralized server vs p2p.

2.2 Manual implementation

Manual implementation of a networking layer for multiplayer games is the most common choice for big development video game studios. Notable examples can be found in games such as Starcraft, Age of Empire, Supreme Commander, and Half-Life. This solution is tightly bound to the behaviour of game entities and requires to implement all the networking components by hand, from the data transmission primitives to the synchronization logic and client-side prediction. In this scenario the data transmission and synchronization will be dependent on the structure and behaviour of the entities. Indeed, the transmission must

take into account the nature of the data describing the game entity (value or reference) and the action it might perform during the game. For instance, let us assume that we want to synchronize the action of shooting with a gun, the game creates a projectile locally when the player presses the button to shoot and that information must be shared with all the players. Furthermore, the modules functionality is not dedicated to a specific task but interconnected with that of other game components. Thus, this solution is affected by two main problems: *tight coupling*, *low cohesion*. Notable examples of these problems were encountered during the development of Starcraft [7], where it was reported that the game code for specific units (including multiplayer behaviours) were branched and completely diverged from the rest of the program, and Half-Life [1], where the transmitted data was bound to a specific pre-defined data structure and any change to the game had to be reflected there. Moreover, the networking code flexibility and the chance of re-use it in different games is non-existent as it is strictly bound to the specific implementation of the game.

2.3 Game Engines

Game engines overcome the problems of *low cohesion* by providing built-in modules and functionalities to manage the network transmissions for multiplayer games. Unity Engine [3] and Unreal Engine [4] are both commercial game engines that support networking. Their approach is centred around a centralized server approach, where the "real" state of the game is stored on the server and the clients only see its local approximation, which must be periodically validated and possibly corrected. The game engine offers the chance of defining what entities should be replicated on the clients and the synchronization is automatically granted by the engine itself, with the possibility of customizing some parameters, such as the update frequency and the prediction methods. Further customization can be achieved by using *Remote procedure calls* (RPC's) by specifying (*i*) that a specific function is called on the server and executed by the clients (for instance, to control events that do not affect gameplay, such as graphical effects), by(*ii*) that a function is called on the client and executed on the server (for instance, to perform actions), and (*iii*) multi-cast calls (executed on all the clients and the server).

The approach based on game engines grants *high cohesion*, because it allows to separate the logic of the game from the logic of the networking layer by providing library-level abstraction. The problem of *tight coupling* persists because the game logic will be still connected to the several networking components of the game engine. This solution also offers little to no code re-usability and portability, since anything related to the networking part (and also in general to the whole game implementation) will be bound to the specific game engine, thus, for instance, migrating from Unity Engine to Unreal Engine would require to rewrite most of the code.

Technique	Loose coupling	High Cohesion	Re-usability/portability	Extensibility
Manual implementaion	✗	✗	✗	✗
Game engine	✗	✓	✗	✗
Compiler	✓	✓	✓	✗

Table 1: Advantages and disadvantages of networking implementation techniques

2.4 Language-level primitives

Another option is to implement primitives at language level, that is to build a programming language to support networking primitives to define network synchronization behaviours and data transmission. This approach grants *high cohesion* and *low coupling*, since the details of the networking are hidden and abstracted away by the language itself and the game will only interface with the language layer. This approach was explored in scientific works such as the *Network Scripting Language* (NSL) [6]. This approach grants also code re-usability and portability, as the networking code is bound only to the language and not to a particular engine or specific implementation of the game. For this reason code related to components of the game that exhibits similar behaviours can be re-used for other games. This approach requires to build a compiler (or interpreter) for the language, which makes extending the networking layer with additional functionalities difficult as any change to the networking language must be reflected in the implementation of the compiler itself.

2.5 Towards an approach based on meta-compilation

.

In the previous considerations we have analysed different techniques to implement network communication for a multiplayer game. In order to evaluate the goodness of each solution we considered four parameters:

- *Loose coupling*: the measure of how closely two modules are.
- *High cohesion*: the degree to which the elements of a module belongs together.
- *Code re-usability and portability*: how it is possible to re-use the code for a different similar game and how much of the code can be ported to this version.
- *Extensibility*: how much it is possible to extend the networking implementation with additional functionality.

3 Networking architecture and semantics

- `master`, `slave rules`, and `connect` rules.
- `send`, `send_reliable`, `receive`, `receive_many`, `let!`
- Formal semantics

4 Implementation in Metacasanova

Implementation in Metacasanova.

5 Evaluation

6 Conclusion

References

[1] Yahn W Bernier. Latency compensating methods in client/server in-game protocol design and optimization. In *Game Developers Conference*, volume 98033, 2001.

[2] Nicolas Ducheneaut, Nicholas Yee, Eric Nickell, and Robert J Moore. Alone together?: exploring the social dynamics of massively multiplayer online games. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 407–416. ACM, 2006.

[3] Unity Game Engine. Unity game engine-official site. *Online][Cited: October 9, 2008.] http://unity3d. com.*

[4] Epic Games. Unreal engine 3. *URL: http://www. unrealtechnology. com/html/technology/ue30. shtml*, 2006.

[5] Lothar Pantel and Lars C Wolf. On the impact of delay on real-time multiplayer games. In *Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*, pages 23–29. ACM, 2002.

[6] George Russell, Alastair F Donaldson, and Paul Sheppard. Tackling online game development problems with a novel network scripting language. In *Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games*, pages 85–90. ACM, 2008.

[7] Patrick Wyatt. The starcraft path-finding hack. `http://www.codeofhonor.com/blog/the-starcraft-path-finding-hack`, 2013.