

Automated networking in Meta-Casanova

Francesco Di Giacomo, Mohamed Abbadi, Giuseppe Maggiore,
Agostino Cortesi, Pieter Spronck

Abstract

Multiplayer support is a desirable feature in commercial games since it allows to extend the longevity of a title. The same importance can be found in several examples of serious games, as they are often used in training where several individuals must cooperate to reach a common goal. Having smaller resources than big entertainment companies, serious games developers cannot afford to develop ad-hoc networking code and need to rely on game engines. On the other hand, game engines require deep knowledge and do not overcome all the problems of hand-made networking implementations. For this reason, it is desirable to express networking behaviours in a dedicated language, or Domain-specific language, that provides the necessary abstractions, but this approach suffers from having to implement a compiler for the language, which is hard to extend. In this paper we provide an alternative approach based on meta-compilation defined in Metacasanova, a meta-compiler where it is possible to specify the formal semantics of the language. We show that using Metacasanova notably reduces the effort of developing a Domain-specific language while keeping the same advantage in terms of quality of code.

1 Introduction

Adding multiplayer support to games is a desirable feature to increase the game popularity by attracting millions of players [5]. 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 [10]. 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, while in a First-person shooter, which gameplay is based on fast-paced action, low latency is paramount. In addition to games developed for the entertainment sector, networking is a crucial feature also in the field of serious games. However, serious games developers do not have access to the same amount of resources and manpower available at big companies such as EA or Ubisoft, while having to deal with the same degree of complexity in their products.

Put some examples of existing serious games

For this reason, manually implementing the networking module is not a feasible solution in the field of serious games, as serious games developers have to rely on game engines, such as Unity 3D or Unreal Engine, which offer a built-in API to deal with networking. Although being a better approach than developing a hand-made networking protocol for a specific game, game engines require extensive knowledge to be used effectively and require, to some extent, to still deal directly with low-level issues related to data transmission and remote procedure calls. This problem can be solved in part by developing a Domain-specific language including abstractions to define the behaviour of the game in a networking scenario. Choosing this approach requires to implement a compiler for the language, which can be argued to be as time-consuming as developing a hand-made networking implementation. In this paper we propose an alternative approach to the development of a hard-coded compiler for a Domain-specific based on meta-compilation, which has been proven to be an effective technique in different works [9, 4], in order to have the benefits of a domain-specific language for networking and, at the same time, a lower development effort.

Complete with summary of the paper

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 measured in terms of four characteristics:

- *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*: the ease of extending the networking implementation with additional functionality.

Finally, we present our alternative solution and formulate the 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 [12], 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 [2], 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 [6] and Unreal Engine [7] 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.

2.4 Language-level primitives

Another option is to implement primitives at language level, that is to build a *domain-specific language* to support networking primitives defining 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) [11] or Casanova Language [8]. 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 analysed different techniques to implement network communication for a multiplayer game. In order to evaluate the goodness of each solution, we evaluated it in terms of loose coupling, high cohesion, code re-usability and portability, and extensibility. Table 1 summarizes this evaluation, pointing out whether a specific technique grants one of the properties. The solution that grants the highest number of properties is the one using abstractions defined in a Domain-specific language. As stated above, this solution still lacks extensibility as all further extensions to the language must be reflected directly in the compiler implementation. Thus, the capability of extending the language is strictly connected with the capability of extending its compiler with additional language constructs. This is known to be a hard and time-consuming task, as a compiler is made of several inter-operating components responsible for the translation steps and the semantics analysis of the program [1, 9]. Regardless of this, the only creative aspect of defining a compiler is not the implementation of such modules, that always follows the same logic, but only the definition of the language constructs [3]. Meta-compilers are programs that are able to take as input the definition of a language, defined in a meta-language, a program written in that language, and generate executable code. Using a meta-compiler to define the semantics of a Domain-specific language has been proven useful in several situations, for example in [9, 4], but to our knowledge never attempted in the scope of creating language-level primitives for multiplayer games development. Given these considerations we formulate the following problem statement:

Problem statement: *Given the formal definition of language-level networking primitives, to what extent using a meta-compiler eases the process of integrating them into an existing Domain-specific language?*

Technique	Loose coupling	High Cohesion	Re-usability/portability	Extensibility
Manual implementaion	\times	\times	\times	\times
Game engine	\times	\checkmark	\times	\times
Language-level primitives	\checkmark	\checkmark	\checkmark	\times

Table 1: Advantages and disadvantages of networking implementation techniques

In what follows, we will present Casanova 2, an existing Domain-specific language, and Metacasanova, a meta-compiler used to give a re-implementation of Casanova 2. We then proceed to present a networking architecture defined in Casanova 2 and we give the semantics of its primitives. We finally show an implementation of such semantics in Metacasanova and we compare the code length with its respective implementation counterpart in the Casanova 2 hard-coded compiler.

3 Networking architecture and semantics

- overview of Casanova 2.
- `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] Alfred V Aho, Ravi Sethi, and Jeffrey D Ullman. *Compilers, Principles, Techniques*. Addison wesley Boston, 1986.
- [2] Yahn W Bernier. Latency compensating methods in client/server in-game protocol design and optimization. In *Game Developers Conference*, volume 98033, 2001.
- [3] Erwin Book, Dewey Val Shorre, and Steven J. Sherman. The cwic/360 system, a compiler for writing and implementing compilers. *SIGPLAN Not.*, 5(6):11–29, June 1970.
- [4] Francesco Di Giacomo, Mohamed Abbadi, Agostino Cortesi, Pieter Spronck, and Giuseppe Maggiore. *Building Game Scripting DSL’s with the Metacasanova Metacompiler*, pages 231–242. Springer International Publishing, Cham, 2017.

- [5] 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.
- [6] Unity Game Engine. Unity game engine-official site. *Online*[Cited: October 9, 2008.] <http://unity3d.com>.
- [7] Epic Games. Unreal engine 3. URL: <http://www.unrealtechnology.com/html/technology/ue30.shtml>, 2006.
- [8] Francesco Di Giacomo, Mohamed Abbadi, Agostino Cortesi, Pieter Spronck, Giulia Costantini, and Giuseppe Maggiore. High performance encapsulation and networking in casanova 2. *Entertainment Computing*, 2017.
- [9] David Kågedal and Peter Fritzson. Generating a modelica compiler from natural semantics specifications. In *Summer Computer Simulation Conference*, pages 299–307, 1998.
- [10] 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.
- [11] 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.
- [12] Patrick Wyatt. The starcraft path-finding hack. <http://www.codeofhonor.com/blog/the-starcraft-path-finding-hack>, 2013.