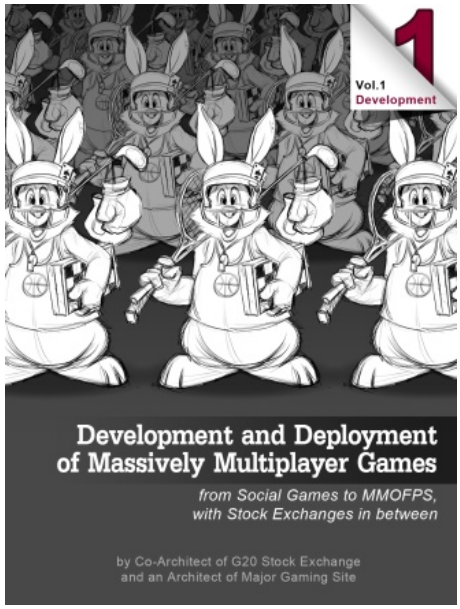
Chapter II: "Game Entities and Interactions" from upcoming book "Development and Deployment of MMOG"

posted November 2, 2015 by "No Bugs" Hare, translated by Sergey Ignatchenko

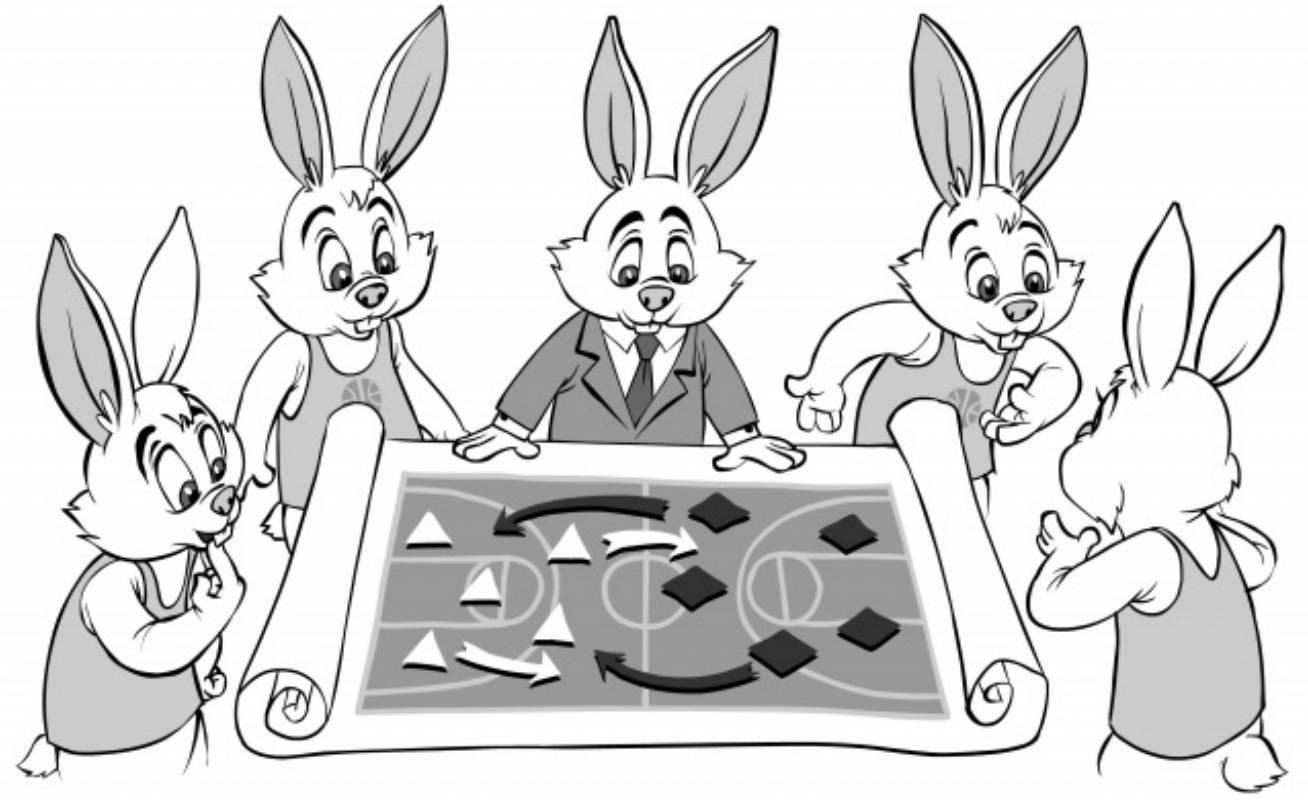
[[*This is Chapter II from the upcoming book "Development&Deployment of Massively Multiplayer Online Games", which is currently being beta-tested. Beta-testing is intended to improve the quality of the book, and provides free e-copy of the "release" book to those who help with improving; for further details see "Book Beta Testing". All the content published during Beta Testing, is subject to change before the book is published.*



Please note that this Chapter II may look boring for some of the developers; don't worry, there will be a lot of code-related stuff starting from Part B, but at the moment we need to describe what we're dealing with.

To navigate through the book, you may want to use Development&Deployment of MMOG: Table of Contents.]]

So, after universally-hated unacceptably-long 2-hour meeting (the one where you discussed your business requirements) you've got your very first idea about what you're going to do. With luck (or if you've read previous Chapter on what you'll need from the network perspective) it contains answers to all the questions we need. Of course, these answers are subject to change, but at least you know what you will be dealing with at the moment. Now, you may think that you know enough to start architecting the game; however, it is not the case (yet). The next step after specifying business requirements is not about drawing an architecture diagram, or (Stroustrup forbid!) a class diagram. This next step should be a thorough understanding of game entities and their interactions.



On Importance of Holding Your Horses

After you've got your Business Requirements, it is often tempting to say "hey, we will be using such-and-such game engine, so all we need is to implement our game around this engine". Or (especially if you're coming from web development) to say the same thing, but building the game around the database. Or to build your game around some protocol (TCP or UDP). However, at this stage you still don't really have sufficient information to make architectural decisions. All these engines, databases, and protocols are nothing more than implementation details, and we're not at the implementing stage yet.

"While your game is likely to have a 3D engine, and very likely to have some DB to provide

While your game is likely to have a 3D engine, and very likely to have some DB to provide persistence, and will certainly need to run on top of some IP-based protocol, it is too early to make any of them a center of your game universe. In particular, even decision whether your game should be game-engine-centric, or 3D-engine-centric, or DB-centric, or protocol-centric, requires more thorough understanding of the game mechanics, that usually arises right after reading Business Requirements.

Making these decisions (and actually any architectural decisions for that matter) before you have Entities&Relations diagram described in this Chapter, can severely restrict your choices, and if you have made a mistake in making such a

persistence, and will certainly need to run on top of some IP-based protocol, it is too early to make any of them a center of your game universe.

decision (and when you're deciding without having sufficient information, mistakes are more than likely), it may easily lead to grossly inefficient and even completely unworkable implementations.

For example, if you decide that "our system should be DB-centric, with 100% of the state being written to DB at all times", and your system happens to be a blackjack site, your implementation will cause about 10x more DB load than an alternative one, plus you will get a bunch of issues with implementing rollback in case if your site has crashed (which causes many games to be interrupted in the middle, and with a multiplayer site you do need some kind of rollback). Usually, the most optimal implementation for many of casino multi-player games is with state of the *table* being stored in-memory only (and synchronized with DB only when one single game is completed), but this won't become obvious until you draw your Entities&Relations diagram.

In an another example, if you decide that "our system should be game-engine-centric", and your game engine of choice doesn't support a concept of *zones* (and doesn't have another way to calculate set of players-who-need-to-know-what-this-player-does, assuming that everybody-interacts-with-everybody instead), you may end up with a system which works reasonably well for small virtual worlds, but which is completely unscalable to larger ones due to O(N^2) traffic which pretty much inevitably arises from everybody-interacts-with-everybody assumption.

TL;DR:

- **DON'T Start with Architecting Around Game Engine**
- **DON'T Start with Architecting Around DB**
- **DON'T Start with Architecting Around Protocol**
- **Overall, it is still too early to start architecting your game.**

Instead,

What you need before architecturing is a diagram showing all the game entities, and all their interactions

Whether this step needs to involve project stakeholders, depends on the nature of the game and on the level of details in your business requirements. However, in any case it is advisable to have project stakeholders available during this stage, as

questions on interactions such as “is entity A allowed to interact with entity B?” are very likely to arise.

Game Entities: What are You Dealing With?

In each and every game, you have some game entities, which you’ll be dealing with. For example, in a MMORPG you’re likely to have PCs, NPCs, zones, and cells; in a casino game you have lobbies, tables, and players; in a social farming game you have players and player farms. Of course, every game will contain many more entities than I’ve mentioned above, but they depend on specifics of your game, so you’re certainly in much better position than myself to write them down. And if you feel that you’re about to be hit by “not seeing forest for the trees” syndrome, you can always replace your diagram with several ones (organized in a hierarchical manner), so that each one contains only a manageable number of entities.

Interactions Between Game Entities

Those game entities from the previous chapter, usually need to interact with each other. Players reside within cells which in turn reside within zones, PCs interact with NPCs, players sit and play on casino tables, and players interact with other player’s farms. All these interactions are very important for the game architecture, and need to be written down as a part of your Entities diagram. Even more importantly, you need to be reasonably sure that you have listed *all* the interactions which you can think of at the moment.

“Even more importantly, you need to be reasonably sure that you have listed *all* the interactions which you can think of at the moment.

What Should You Get? Entities&Interactions Diagram

As a result of the process of identifying your game entities, you should get a diagram (let’s name it “Entities&Interactions Diagram”) showing all the major game entities and, even more importantly, *all possible interactions* between these entities.

One thing which MUST be included into the Entities&Interactions Diagram (alongside with gameplay-related entities), is entities related to monetization (payments, promotions) *and* entities related to social interactions. In other words, if you’re going to rely on viral marketing via social networks, better know about it in advance; as discussed below, the impact of social interactions on architecture can be much more significant and devastating than simple “we’ll add that Facebook gateway later”.

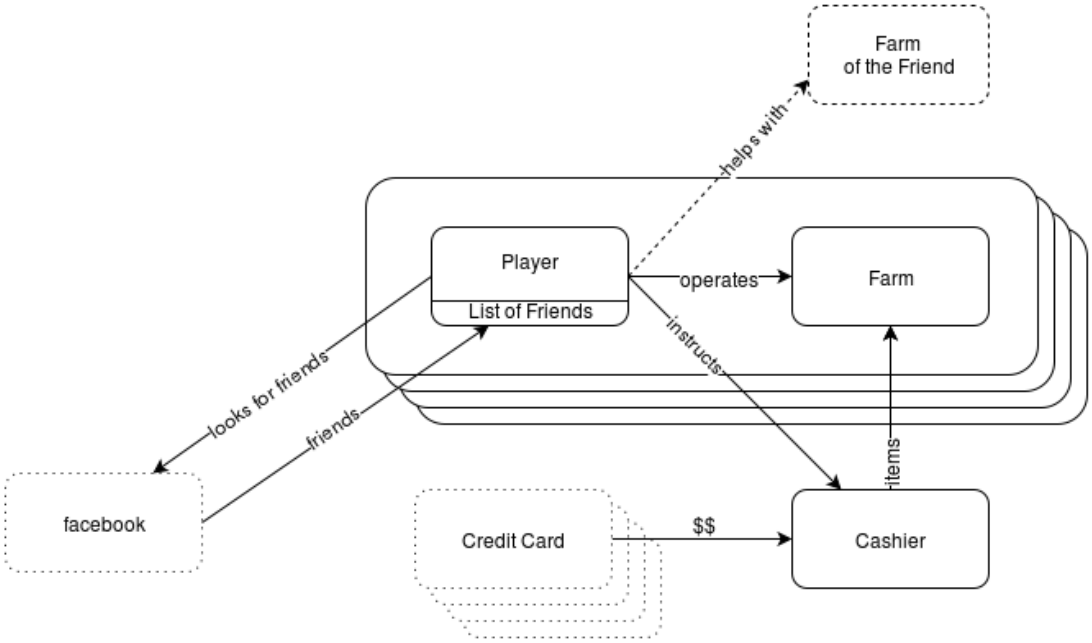
Examples of Entities and Interactions

To give you a bit of idea on entities and interactions, I'll try to describe typical entities for some of popular game genres. Note that (as with any other advice, in this book or elsewhere), *YOUR MILEAGE MAY VARY, and you need to think about specifics of your game rather than blindly copying typical entities mentioned below! Also note that example diagrams provided here illustrate only a few aspects of the game; in practice, your diagrams will usually be much more complicated.*

Social Farming and Farming-Like Games

While social games genre is very wide and is difficult to generalize, one sub-genre, social farming games, is straightforward enough to describe. In farming and farming-like games number of different entities and especially interactions between them are quite limited. Entities are usually limited to *players* and their *farms* (the latter including everything-which-can-be-found-on-the-farm). Interactions (beyond *player* interacting with their own *farm*) are also traditionally very limited (though they are important from the social point of view).

Fig II.1



NB: On all our example Entities&Interactions Diagrams, we will draw external (to our game) entities as dotted;[1] *feel free to use any other convention, the idea here is not to create yet another formal-and-unusable diagram language, but for you to visualize what your game is about.*

You should keep in mind that in most cases there is one significant caveat to remember about: it is a mistake to think that you can arbitrarily separate players onto different servers and allow only interactions within one such server. This "interactions are allowed only within one player server" model would work only until you introduce "Play with your Facebook

friends" feature (which will most likely be a Business Requirement, if not now, then a little bit later, see more on it in "On Arbitrary Player Separation" section below). And introducing "play with real-world friends" concept affects arbitrarily separated player servers dramatically: you no longer know which players want to play together, and inter-server interaction is no longer non-existent (and in fact the inter-server interaction has been observed to cause severe problems to some of the social games).

Overall, you should not rely on arbitrary player separation, even for social games

> "Introducing "play with real-world friends" concept affects arbitrarily separated player servers dramatically: you no longer know which players want to play together, and inter-server interaction is no longer non-existent.

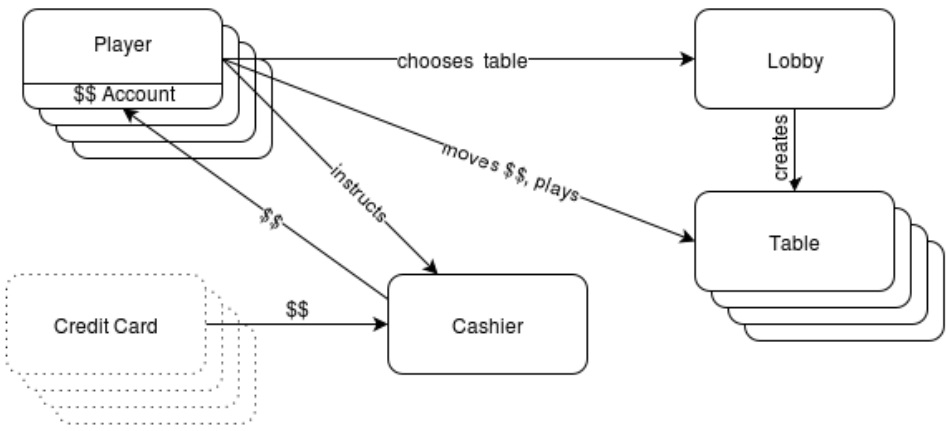
---

[1] at architecture stage, we'll need to insert appropriate gateways to communicate with these external entities, but we're not there yet

Casino Multiplayer Games

With casino multiplayer games, everything looks quite simple: there are *tables* and *players* at these tables. However, in some of the casino games (notably in poker), choosing an opponent is considered a skill, and therefore players should be able to choose who they want to play against. It implies another game entity – lobby, where the opponents can be selected. An example Entities&Interactions Diagram for Multiplayer Blackjack is shown on Fig II.2:

Fig II.2



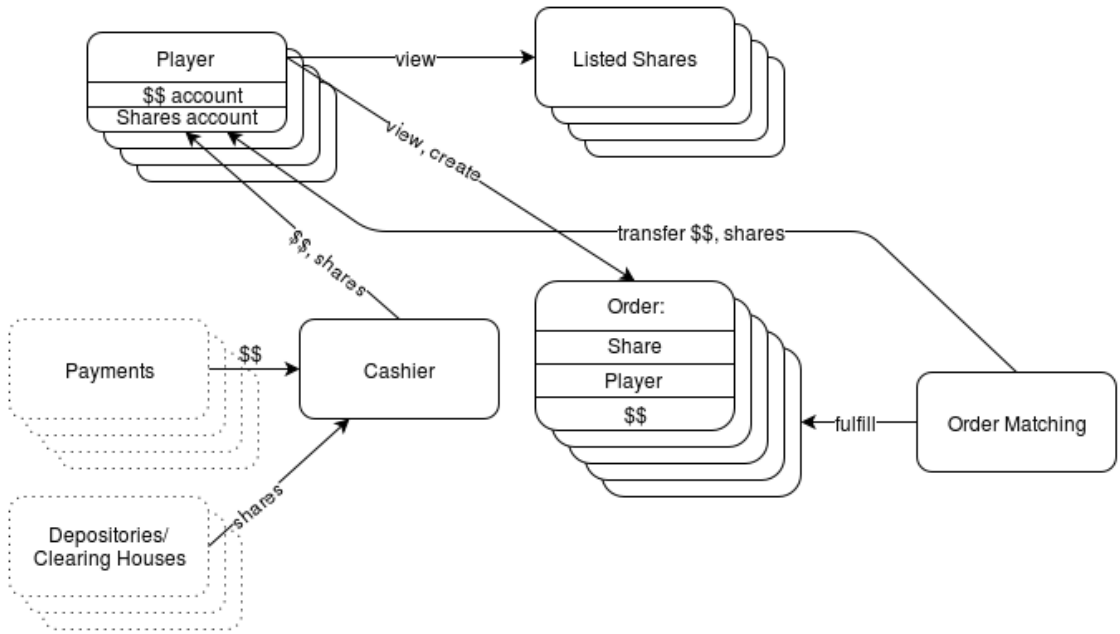
Note that for this example diagram, we've omitted social interaction; you will need to add it yourself, as it is appropriate for your specific game.

Stock Exchanges, Sports Betting and Auction Sites

In fact, stock exchanges and auction sites are so close to betting, that you'll be facing significant difficulties when trying to describe the difference between the three (except, obviously, for social stigma traditionally attached to betting). With stock exchanges, auction sites (think "eBay") and betting sites, entities involved are the same. It is *players* (though, of course, for a stock exchange you need to describe them as "traders" or "dealers"), and *stocks* (or sporting events/products). *Players* don't interact directly; however, indirect interaction does exist via creating some actions ("orders" or "bets") related to stocks or events/products.

Fig II.3 shows an example Entities&Interactions diagram for a stock exchange:

Fig II.3



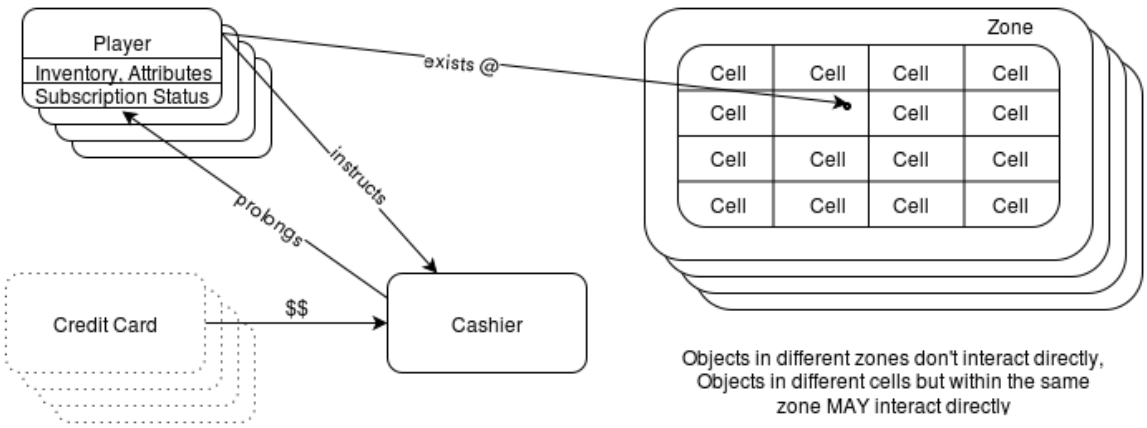
Virtual World Games (MMOTBS/MMORTS/MMORPG/MMOFPS)

Despite all the differences (including those which affect architecture a lot, as it will be described in Chapter [[TODO]]), from the point of view of the entities involved all the virtual world games tend to be more or less similar. In particular, in these games there are players (*PCs*), there are *NPCs*; also there are usually *cells* and *zones* containing those cells, which represent a virtual world (VW) where interactions between PCs and NPCs are occurring. What is important is that players in separate *cells* usually can interact, but players in different *zones* cannot.[2] The player option of choosing who she wants to play with, may or may not be provided; however, even if it is not provided, and you think that you can toss your players around your virtual worlds as you wish, arbitrary player separation (assigning player to servers without any inter-server interaction) becomes infeasible as soon as you introduce a

social feature such as “Recruit a Friend”. More on arbitrary player separation in “On Arbitrary Player Separation” section below.

Fig. II.4 shows an example Entities&Interactions Diagram for an MMORPG:

Fig II.4



[2] terms *cells* and *zones* may vary depending on the engine, but the idea is usually pretty much the same

On Arbitrary Player Separation

As we’ll see in Chapter [[TODO]], implementation-wise it is often very tempting to consider all your players as commodity, and to think that you can arbitrary assign players to servers without any communication between those servers (one of the things I’ve seen, was relying on limited “socializing” within a single server). However, for vast majority of games out there, it is a really bad idea. In short: don’t do it.

IRL
Abbreviation for 'In Real Life.' Often used in internet chat rooms to let people you are talking about something in the real world

The reason behind is the following. Even if your game as such seems to allow such arbitrary player separation (without any interaction between player servers), there are Big Fat Chances that you will need to implement some kind of interaction between players sooner rather than later. Pretty much any kind of IRL integration requires some kind of interaction between players just because they want it (and not because your rule engine decided that these two guys belong to the same server). In other words: if you don’t think that interaction of players just-because-they-want-it is a Business Requirement – think again.

> **and not in the internet world.**
> — *Urban Dictionary*

As just one simple example, even a simple "Play with your Facebook friend" feature requires players to "know" about each other, and to interact with each other. Moreover, you cannot possibly predict in advance which of your players will form a Facebook friendship some months later. In an another simple example, most of payment processing will require you to have some cross-server analysis to prevent fraudsters-who-cheated-one-server to cheat on another one. And so on. And so forth. And we didn't even start to speak about support people, who will need to manage all this mess. Trying to ignore this IRL integration is a pretty much sure way to a post-deployment disaster.

Think of your game as of one single planet with bunches of players living their, not as of cluster of non-interacting asteroids with a few players on each. As a rule of thumb, this stands even for such seemingly unconnected games as farming: as soon as you add some socializing (and it is usually a very important part of business strategy), you do need inter-player interactions, with no ability to control which players are communicating with each other.

Entities&Relations Diagram as a Starting Point to Architect Your Game

This Entities&Interactions Diagram you've just got is one of those things which will affect your architecture greatly. In particular, it is a starting point to realize what kinds of "implementation entities" (such as *servers, OS processes, DB tables, rows, and columns, etc.)* you need to implement your "game entities", and how to map game entities to implementation entities. This "how to map game entities into implementation entities" question will be discussed in Chapter [[TODO]].

[[To Be Continued…

This concludes beta Chapter II from the upcoming book "Development and Deployment of Massively Multiplayer Games (from social games to MMOFPS, with social games in between)". Stay tuned for beta Chapter III]]

EDIT: beta Chapter III. On Cheating, P2P, and [non-]Authoritative Servers, has been published.

Acknowledgement

Cartoons by Sergey Gordeev from Gordeev Animation Graphics, Prague.

« *Chapter I: "Business Requirements" from upcoming book "Dev*.

Chapter III. On Cheating, P2P, and [non-]Authoritative Server.. »

Filed Under: Distributed Systems, Network Programming, Programming, System Architecture
Tagged With: game, multi-player