

Everything but the Game

Robert Mitchell
Sony Online Entertainment
ogdc2007@mountabbey.com

Abstract

Programming an online game consists working on the game and everything but the game. This paper will ignore the first half and explore some of the details about the second. The areas covered include patching, chat, persistent messaging, localization, user authentication, billing, customer support tools, logging, monitoring, security, testing/debug tools, community tools, interactive web and mobile extensions, auctions, and match making. The objective is to reveal some of the issues and choices that have to be made when adding these features to an online game. The motivation is that often overlooked and forgotten components will differentiate your service from the rest of the games.

1. Exposition

For our purposes, “everything but the game” can be generalized as game related systems that are not driven by input from game designers or artists. They are motivated by the nature of the online space that makes community and social networks a critical element of a project’s success. They are motivated by the service nature of the worlds we are creating. And “everything but the game” is driven by business and economic realities.

We will start by trying to break down everything into three categories: game, not game, and not covered. And then, we will walk through the items and discuss why each is important and some implementation issues. I hope to justify the engineering time spent on these features and motivate new projects to plan beyond just the game.

1.1. *Game, not game, or not covered*

I propose that we try to separate everything involved in the development and maintenance of an online game into three categories: game, not game, and not covered. The contents of these categories may appear somewhat arbitrary and depending on the technology/business paths a game and company take, the placement may look slightly different.

The distinctions I would like to make are that the category "game" is for systems that are required to ship the game. "Not covered" are things that likely fall outside of the game teams areas of responsibility. And to stick to the title of the paper, use "everything but the game" for everything else.

1.2. *Game*

While every game is different there are some game development tasks that we can probably all agree should be grouped into the game category. These things are required for a game to ship.

The primary set of systems includes things whose creative design is controlled by game designers and artists. There is probably little doubt these are “game”.

- Art and audio: environments, models, animation, textures, images, sounds, particles.
- Game play systems: combat, quests, items, puzzles, crafting, collecting, mini-games, player versus player (PvP), scoring
- Story and lore: background fiction and everything used to convey that to the player
- The second set is the traditional realm of game programming with the obligatory online and persistent components included.
- Tools: to generate all of the above
- Client: rendering, user interface, and game presentation
- Artificial intelligence: non-player character interaction with the player
- Server code: architecture, multiplayer support
- Database: persistence

And depending on the game and how broadly you interpret the categories, the game will include more, lots more.

1.3. *Not covered*

I initially didn’t want to have this category. However, there is already enough to discuss without trying to tackle everything necessary to create, ship, and maintain an online game. And so some systems just need to be ignored.

- The right way: that’s left as an exercise for the reader
- Installers: outside the game
- Process for <insert task>: from the tactical questions like how often and when, to the practical operations...ignored
- Network and server management: certainly some of the game engineering will help monitor and react to requests but someone else will be able to cover this in much more detail.
- Customer service policies: similar
- Marketing and sales: important but outside the game
- Legal, accounting, and everything else the publisher does: not game

1.4. *And everything but the game*

And that brings us to the category we are most interested with for this discussion. Very few of these systems are necessary for a single player game. And nearly every one of them would need to be considered if you were building an interactive online community.

- Patcher and a way to update the game and data
- Chat and persistent messages
- Localization and Internationalization
- User authentication
- Billing
- Customer support tools
- Logging and Monitoring
- Security
- Testing/Debugging tools
- Community tools
- Web and Mobile extensions
- Match making
- Auctions and Microtransactions

1.5. *It's not a game*

Having categorized systems as game, not game, and not covered, I'd like to take a step back and explain why any of this matters.

One reason is that online games are not the short lived computer games there were previously created. Not many computers are still useful eight years later. But if you looked, you'd be able to find examples of online games that have been active at least that long.

Depending on the revenue model, the distribution, and the publisher, a game might not just be a game. It may be content to drive advertising sales. It may be an attraction in a greater virtual theme park. It may be the central square of a virtual community. The game may just be part of a collection of media, building exposure for the underlying intellection property. And that's not to say the game can't be something else entirely.

Not much of that is relevant to the players though. To them, an online game is a place to be entertained, to discover and to express themselves, and to hang out with friends. These features are here to help them build and maintain these friendships and to make it a place where they choose to keep coming back.

Precursor and disclaimer

The remainder of the paper presents a flavor of each of the items in the "everything but the game" cluster. Each of the topics in the following sections would be suitable for an entire article or two. I can only hope to provide an introduction and show some of the issues that should be considered before implementation and some of the potential pitfalls. Your requirements, constraints, and business objectives may be significantly different. So use this

information as the beginning of the discussion and not as a complete compilation of the data. Also, it is important to note that as online games mature, additional features may become part of the players' expectations.

1.6. *Patching*

An online game suggests the ability to update the game, to fix bugs and/or to add features. I consider this ability fundamental.

A console only developer might taunt me because this implies an unfinished game when it ships. Indeed, an online game by its nature is never finished. Instead, I prefer to think of patching as the ability to add additional functionality and content, as adding challenges and removing unnecessary obstacles for the players, and as extending the life of a game.

With every current generation console including networking capabilities, expect all of the "everything but the game" systems to start applying on those platforms too.

Another benefit to having a patching mechanism is that making sure that the latest version of the client is running may be necessary to prevent exploits and other security issues.

Patching the client

The traditional patching discussion involved the client. You will want to be able to send updated data and code to the client. Depending on the client type, this ability might be explicit (Flash or Java applet). A poor man's implementation of a patcher is to have the user re-download the entire game. Obviously, smarter choices are available.

Some potential patching implementations include using HTTP Get, BitTorrent, and a revision control API like SubVersion. If you decide to implement your own solution, using compression and delta encoding will reduce the bandwidth overhead of the patches.

Staging Client Data

Staging client data ahead of the anticipated update reduces the downtime. One example implementation is the Background Intelligent Transfer Service (BITS) extension to Windows. BITS automatically detects network activity and only downloads the data when the connection would otherwise be idle.

A game can minimize downtime by staging the new code and data beforehand. I would like to see teams take reducing downtime more seriously and suggest that supporting two versions of the game network protocol would be one way to allow for rolling upgrades.

Patching the server

Potentially in the realm of operations and not development, any online game will want to be able to update its servers. Patching the server requires a lot of communication across a lot of groups. Don't forget to implement a system for telling your players.

The process becomes more complicated if you are required to patch different regions at different times or simultaneously supporting different code bases with different publishers.

Ability to toggle features on the fly

While not exactly patching, including the ability to toggle features on the fly both on the client and server will prove a useful addition. A theme park wouldn't want to have to shutdown completely just because one ride was needed maintenance or was malfunctioning and had to be shutdown. Likewise, being able to flag features as enabled/disabled reduces the risk when adding new features. In addition, toggles for content would be useful for holiday events and other content intended to be available shorter than the game's average patch cycle.

1.7. Chat

Community is built around the ability to communicate.

Many ways

Chat has been implemented in a variety of ways. Some games only let players pick from pre-selected text (mostly for concerns about privacy and security for children). Chat can be player to player only, as if spoken (radius around speaker), to their group, to an organization, to the virtual zone, to the server/world, and even to the outside world (via instant messenger interfaces). Various solutions would allow a player to talk to any other player on another server or even in any other game. Some games have the keyboard set up to chat by default while others require you to enter a mode to chat.

The game design aspects of chat are legitimate concerns. And in many cases, there are probably better ways than /shout, /auction, and other channels folks like to spam.

Good grief

The easier it is to communicate, the easier it is to grief other players. Any chat system should have spam and profanity filters in addition to the ability to ignore another player. Robust mechanisms to help players manage chat and their interactions with other players will go a long way towards reducing this form of grief. You may want to make sure that players cannot ignore your support staff though.

Expect to have to restrict chat channels if you have a PvP or offer free trials. Making game play decisions based on the security of your channels may seem reasonable; but assume people will use outside means of communication to bypass whatever restrictions you include.

Lawful intercept and Snooping

Customer support will be discussed later but any section on griefing needs to include some tips on tracking. Lawful intercept is the new speak term for phone tapping. Most chat systems will include some ability for customer service to monitor player conversations. One such system allows an account to be flagged and all messages to and from the player get logged and routed to the game's investigation department.

Voice and Video chat

The day is coming soon when these features will be as common as typing is today.

1.8. Persistent messages

If another player is not online, sending a chat message won't get through. Many games are starting to include means for sending email messages within the game to other players or groups of players. Some allow items and coins to be sent as attachments. If getting access to the mailbox is easy, these messages can be used by the publisher to update players about the game world.

Many of the chat considerations apply. You should expect grievers to try to exploit the mail system, and thus some sort of spam filters and storage limitation are likely requirements.

1.9. Localization and Internationalization

Expect your game to have players from around the world. Plan ahead for the day your game will be distributed in another language. Don't even think about leaving this until the end of the schedule or worse until the game has been shipped.

Localization (L10n) refers to the process of making a game ready for a specific market. Internationalization (i18n) refers to the process of supporting more than one language.

String replacement techniques for different languages may be sufficient for most games. However, sophisticated grammar engines may be required. I recently discovered that Russian names require declension – meaning the character name needs to understand its context to be displayed properly. That's the type of detail that will make your game stand out in its secondary markets.

Localization requires more than just translating the game text. Different regions have different date, time, and number formatting. Your default monetary system and units of measurement may not be familiar to people elsewhere. And sorted lists may need to be resorted. And that's just the beginning if you really want to address the regional differences that players expect.

1.10. User authentication

If a game is going to have any kind of persistent element, there will need to be a way for the user to login. User authentication does not need to be limited to just a Boolean access state: the resulting login can return access levels, unlocked features and expansions, and much more.

Consider allowing the user to have the option of having their public name different than the login name. This keeps the login/password pair more secure.

A couple of customer service issues will need to be addressed if you require users to login. The first is the fact that users will forget their passwords. An automated way of recovering passwords that isn't open to exploitation will be necessary. The second problem is that many users will share their accounts with friends and then ask for assistance when one of the secondary user does something mischievous – like sell all their items or delete their characters.

Passwords are not the most secure and can often be cracked without much effort. With some financial institutes going to SecurID and other physical hardware login solutions, it would not surprise me if we see this form of security for online games.

Parental Controls

One benefit of authenticating the user is that the game can impose parental controls. These may be anything from different content based on age to restricting the duration of play sessions. As the age of your player skews younger, expect more demand from parents for more control.

1.11. Billing

Everyone wants to get paid. But I don't want to get paid to have to think about billing issues. I considered putting this topic in the not covered category except for how essential it is to a developer's ultimate success.

There are enough complications that you should very likely want to consider buying a solution instead of building your own – even if you plan on publishing multiple games. Fraud, taxes, and alternate billing methods besides credit cards should be enough justification for the buy decision. But before you pick a solution, consider expansions and flags for specific features and multiple trial subscriptions and bundles and customer support and reporting and all your specific business requirements.

1.12. Transfer tools

If you have persistent player characters and more than one server, engineer for the ability to transfer characters between them because at some point you will need to do just that. Some of the reasons for wanting to move characters include: to debug live players on a test server, to balance server populations, to merge low population servers, and as a value-added service. The challenge is not so much the primary technical task but all the ancillary concerns of removing all aspects of the player from one server and the social issues.

If you allow players to migrate between servers, be conscious of the two communities involved. Maintaining rules like restricting transfer to similar servers and maximum transferable assets will help minimize the potential disruption.

1.13. Customer support tools

With privilege comes responsibility. At some point, you will have to grant abilities to customer support that can be used for both the benefit and harm of your players. One instance of unfairness or favoritism can damage the reputation of your game. Trust and accountability (logging) are not mutually exclusive.

If a player can do it...

A fairly decent rule of thumb is to give your customer support team the ability to do and undo anything a player can do in the game to herself, without the player having to be online. A player is bound to regret some choice or report some bug which has a simple solution if you could only undo/redo some action. Giving customer service this ability will let them focus on researching the facts of the situation instead of worrying about potential solutions once they discover what happened.

This premise carries over to player organizations like guilds too. If you make sure that nothing is a non-reversible decision, there is a lot less chance of a policy being perceived as arbitrary rewards

and punishments. But more importantly, you will have anticipated the needs of your players.

A critical assumption is that a game can determine what a player has done. Logs are essential and get their own section.

Reporting issues

The player should have the ability to let the game team know about bugs, to ask for support, and to report abuse, griefers, and cheats. The ideal mechanism is seamlessly integrated into the game client.

An extension of this idea is to provide a knowledge base about the game to your players, and potentially a more detailed one for the customer service staff. If every unique question has its answer recorded and every petition has its solution recorded, the players will soon be able to solve their own issues (at least the ones that don't require intervention.)

1.14. Logging

Log everything of value that a player does. Parse the data, mine it, summarize it, and track it. The secrets about your game and what the players are doing are hidden there. Having good logs is the difference between speculation and knowledge.

Logs tend to exist for two primary reasons: figuring out what the player did and figuring out what the code did. In most cases, you'll always want the first type turned on. And for critical bits of code, you may want the reassurance of some logging always on. However for debugging code, add the ability to control the volume and focus of the output and only enable what you need to solve a particular problem. Unfortunately, whatever it is you are not logging is almost inevitably the one piece of data you need. So always enable and monitor automated logging of crashes and other outliers and exceptions.

1.15. Monitoring

My definition of monitoring is real time evaluation and control of the game. Watch load and populations and anything else that starts to be predictive of failures. The anything else part implies correlating log events with failures and adding additional data to the monitor list until nothing else needs to be added.

Monitoring also means tracking your economy, player level progression, and revenues. The health of your game is more than just the server status.

1.16. Security Issues

Expect a portion of the players to search for and find any flaws and loopholes you have exposed. Someone will want to automate your game; someone will want to artificially increase their achievements; someone will want to make money off your game. The only realistic hope a game developer has is to avoid making the hacker's task simple.

A risk analysis will show that the game design and implementation itself will have the holes. Some of your players will likely know more about your game design and the implications than the design team. Use your logs to figure out

what the top players are doing. What your logs for players that jump in level or wealth – and investigate what they are doing.

The client is the next most vulnerable piece of an online game. The data sent to and from the client is next. If you send it to the client, treat it as public knowledge. If you get it from the client, treat it as potentially false or bogus data.

Your players are also vulnerable. Once your security is good enough, it may be easier to lie, cheat, and steal directly from other players. Getting a password through social engineering or a hidden keystroke logger on a compromised computer are common ways of compromising your player's security that you have little or no control over.

1.17. *Testing and debugging tools*

Designers will want to monitor their systems; QA will want to test and retest. The programmers will want to be able to duplicate situations. Spending time on testing tools can have large returns.

Automated testing for large online games is necessary because there can be too much to test between update cycles. Load testing for multiplayer games seems like an obvious necessity. But both of these efforts require an engineering commitment.

Some relatively simple tools, like the ability to create a group or guild without requiring multiple clients can help the testing and debugging process. Adding the ability for a client to setup scenarios and to do anything without actually having to go through the process of doing it will make testing easier. However, don't always rely on faked paths through the code, otherwise you may not be able to verify actual bugs.

In addition, building some way to replay what happened is the beginning of a testing system that can ask what if. And then your debugging tools start becoming part of the design tools and designers can iterate more and more fun out of a particular system or puzzle.

1.18. *Community tools*

The community is the heart and the glue behind a game. Anything you can do to help your community is likely a good thing.

Fan sites have historically been built up around games. Players share their experiences, tutorials, and increasingly collected knowledge about the game. The mod community has continued to expand and many multiplayer games have web sites dedicated to maps, user interfaces, and the shared game experience.

Many if not most games provide a forum or meeting place for players to communicate outside the game. The game may publish leader boards and achievements along with player information. Guilds have their own websites and forums but games are increasingly catering to these players and giving them the ability to host web page, images, and automatically updated information.

The push will be to add social network and web2.0 stuff to game web pages. I expect to see better integration with blogs and even game video to web publishing soon. The challenge will be for developers to decide what features help sell the game and retain their customers and which features are not cost effective.

1.19. *Web and mobile extensions*

In order to stand out from other games, developers have integrated game elements on the web and on mobile phones. And I suspect there will be more interactive maps, interactions with player run vendors, communication with friends and guilds, mini-games, and other micromanagement tasks like inventory management. Some other likely examples include event calendars and reminders. If the game was the center of your social life and free-time, what would your expectations be?

1.20. *Match making*

Not to discount traditional match making, but there are games that need something more than a lobby to help users find an opponent. And strangely enough, the MMO may have a need for match making.

Player versus player games with arenas may have match making lobbies. Multiplayer action games may have the feature to allow players to get into the action immediately. And the looking for group feature might easily be replaced with a more automated means.

For ranked games, an automated match making algorithm may become subject to manipulation. Go back and re-read the section on security.

1.21. *Auctions and microtransactions*

While developer supported real money transfers like the SOE Exchange have been introduced with suspicion and mistrust by some players, the assumptions about the monthly subscription being the only financial model is beginning to be challenged by microtransaction based games. And free games supported by advertising revenue are becoming common again with improving quality. This shift may lead to more flexibility in the business models available to a game.

However, don't assume that the engineering challenges of billing go away. Rather, these alternative revenue models only add to the billing and infrastructure challenges.

2. **Conclusion**

I hope the previous discussion has convinced you that you want to include one or more of these features. The next obvious question is one of implementation. Build it? Buy it? Many of these features may already be available as off-the-shelf components.

Many publishers will require that you use their APIs for many of these features. That is great. If you hope to distribute your game with different publishers, there unfortunately may not be much common between their interfaces or requirements.

Remember to think about the future. Player expectations change, the context in which your game is played changes, the market place changes, and thus so will the list of these ancillary features.

And now, go worry about the game. Otherwise, none of this is even necessary.