Survey of software architecture V0.1

Name of system: Thousand Parsec (TP)
Reviewer: Chun Liu
Date: Oct, 5th, 2011
Author of software: Tim "mithro" Ansell, and main contributor: Lee "llnz" Begg
Author of book chapter: Alan Laudicina, Aaron Mavrinac
Five star rating of book chapter: 3 stars (talks too much on ruleset design and how to play TP)

Purpose of system:

Thousand Parsec is a free and open source project, and its goal is creating a framework for turn-based space empire building games.

Most of the time Thousand Parsec is a framework for turn-based 4X games, which the main game phases contained eXplore, eXpand, eXploit, and eXterminate. Players send scouts across a map and discover new territories (explore).Then, they find a satisfy territory and claim their new territory by creating new settlements (expand). After claimed, they start gathering and use resources in their new territories (exploit). Finally, they minimize the number of rival players by forces or by agreements (Exterminate).

Players can develop their own user interface (UI). Each UI can have its own style. UI can be Graphic 2D, Graphic 3D, web-based, and even text-based.

Players can also develop their own game rules (ruleset). Each ruleset can have different empire-building commands, and each ruleset can have a different victory condition. For example, the victory condition can be the player who explores all the planets wins, or the victory condition can be the player who gathers the most resource in 500 turns wins.

UI is implemented in client side, and ruleset is implemented in server side. The clients communicate with the servers using the Thousand Parsec protocol. Everything talks the same protocol. Therefore, a client can connect to multiple servers, and a server can have multiple clients.

Players can share their UIs and rulesets to each other. Unfortunately, Developing new UIs and rulesets takes a lot of works. The developers need to have experience on C++ or Python programming, since the basics of UIs and rulesets are implemented in C++ or Python.

Basic metrics

KLOC:

In August 2011, 359,895 LOC in total (242,084 Code, 59,238 Comments, 58,573 Blanks)

In the year 2006, the Ohloh project has produced 95 KLOC, while Thousand Parsec's own code count puts it at 90 KLOC.

Project start-up:

January 2002 by Tim Ansell.

Number of major releases:

Protocol releases: TP01 (year 2003), TP02 (year 2004), TP03 (year 2007), TP04 (developing)

Client and Server releases are varied:

Major client:

tpclient-pywx (a 2D client, by Tim Ansell, first version in 2007, current version v0.3.2)

tpclient-pyogre (a 3D client, by Eugene Tan, first version in 2008, current version v0.0.2)

Major server:

Tpserver-cpp (by Lee Begg, first version in 2005, current version v0.7.0)

Number of developers:

47

Size of user community or number of installations:

Size of user community: 11+

Installations: unknown [INC]. Protocol, client, server downloads are separated.

Major stakeholders:

Author: Tim "mithro" Ansell, Major contributor: Lee "llnz" Begg, and other contributors.

Use of concurrency:

Each client and server is varied.

Usually system calls on network layer, and system calls on graphic interface.

Implementation language: (excluding comments and blanks)

Python:	32%
Java:	20%
C++:	19%
XML:	8%
PHP:	7%
C:	6%
Other:	8%

Supporting software:

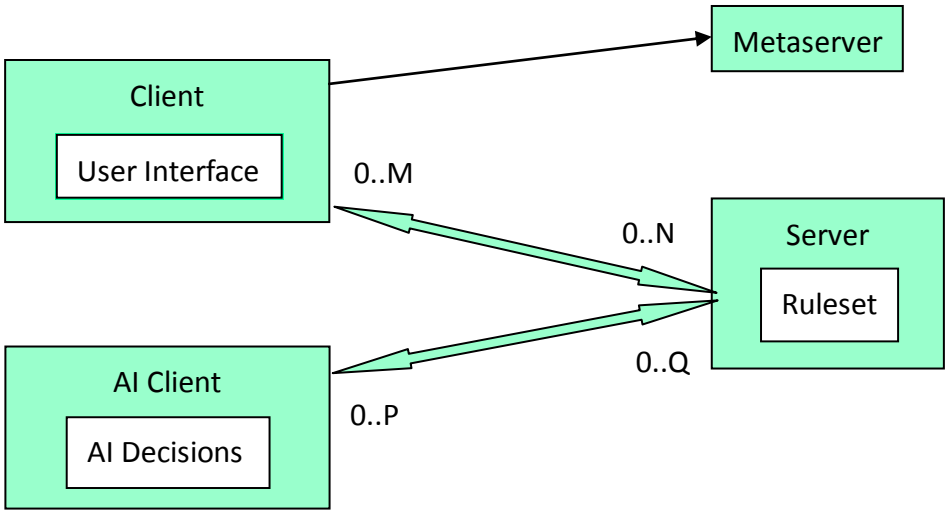
Python Libraries.

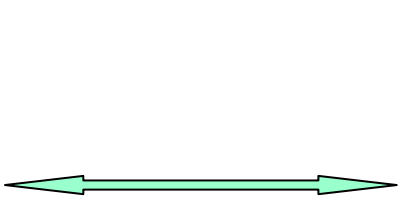
Each client and server is varied. (GNU gettext, MySQL, SQLite, XML …)

High level architecture

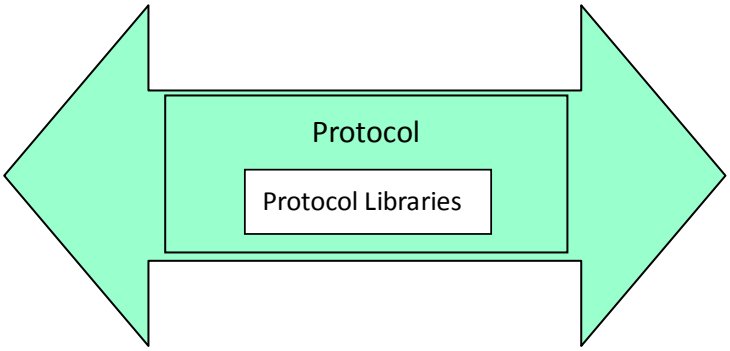
Diagram of software architecturex

(Based on http://www.thousandparsec.net/wiki/Architecture)





means



Means “Component”

Means “Module”

High level scenarios

Before a client requests connection to a server, the client searches on metaserver and finds available servers. Then the client requests connection to a server via protocol. If connection fails, protocol reports fail status to the client. If connection successes, then the game begins.

AI client behaves the same as player's client.

A client sends "request end turn" to server when the client finishes the empire-building commands. There is a time limit at each of the game turn. When the time is up, the server using the ruleset calculates the next state of each client, and sends the result back to clients. When Victory condition reaches, the game ends.

Data structures or algorithms

Protocol:

- Client string: to identify the client.
- Logger class: to log debug, info, warning and error messages.
- GameStatusListener: to get information out of the Protocol Library. Provides notification when the status changes, when connected, redirected, disconnected etc.
- EventLoop
- CacheMethod

Each client side and server side is varied.

Control flow and/or data key to the architecture if any

Protocol:

- Two-Layer structure: ProtocolLayer and GameLayer.
  - ProtocolLayer: handles the low-level network input and output.
  - GameLayer: uses ProtocoLayer to provide resources, messages to client and server.

Each client side and server side is varied.

Architectural style:

Client-Server

Major evolutionary changes:

In project startup, the basic components are Client, Server, and Protocol.

Later, they add AI Client and Metaserver.

Performance bottlenecks:

Protocol (if protocol fails, the game will stop.)

Real time:

Server Side (ruleset calculation)

Notation for architecture:

UML

Methodology:

Looks like 'Agile'. (Google Summer of Code. "a project done in 3 months.")

Appendix

Kruchten's eight context attributes applied to Brown/Wilson systems

1. **Size:** M	(200 KLOC – 300 KLOC exclude comments & blanks)
2. **Criticality:** Lo	(Game)
3. **Age of system:** L	(2002 – present)
4. **Rate of change:** Hi	
5. **Business model:** open source.	
6. **Stable architecture:** Lo	(main architecture is not changed)
7. **Team distribution:** VH	(on ohloh forum project)
8. **Governance:** Lo	(no regulation)