Survey of software architecture V0.0

See book: http://www.aosabook.org/en/index.html

Name of system: Asterisk
Reviewer: Alan Kinzie
Date: October 31st, 2011
Author of software: Mark Spencer
Author of book chapter: Russell Bryant
Five star rating of book chapter: ****
Purpose of system:

The purpose of Asterisk is to be a phone system. It was created because Mark Spencer's company (Linux Support Systems, at the time) did not have enough money to spend on a phone system, but needed one nonetheless. Asterisk was designed to be flexible in the types of technology it can connect – as long as there is a channel driver built for a protocol, asterisk can handle a protocol.

Basic metrics

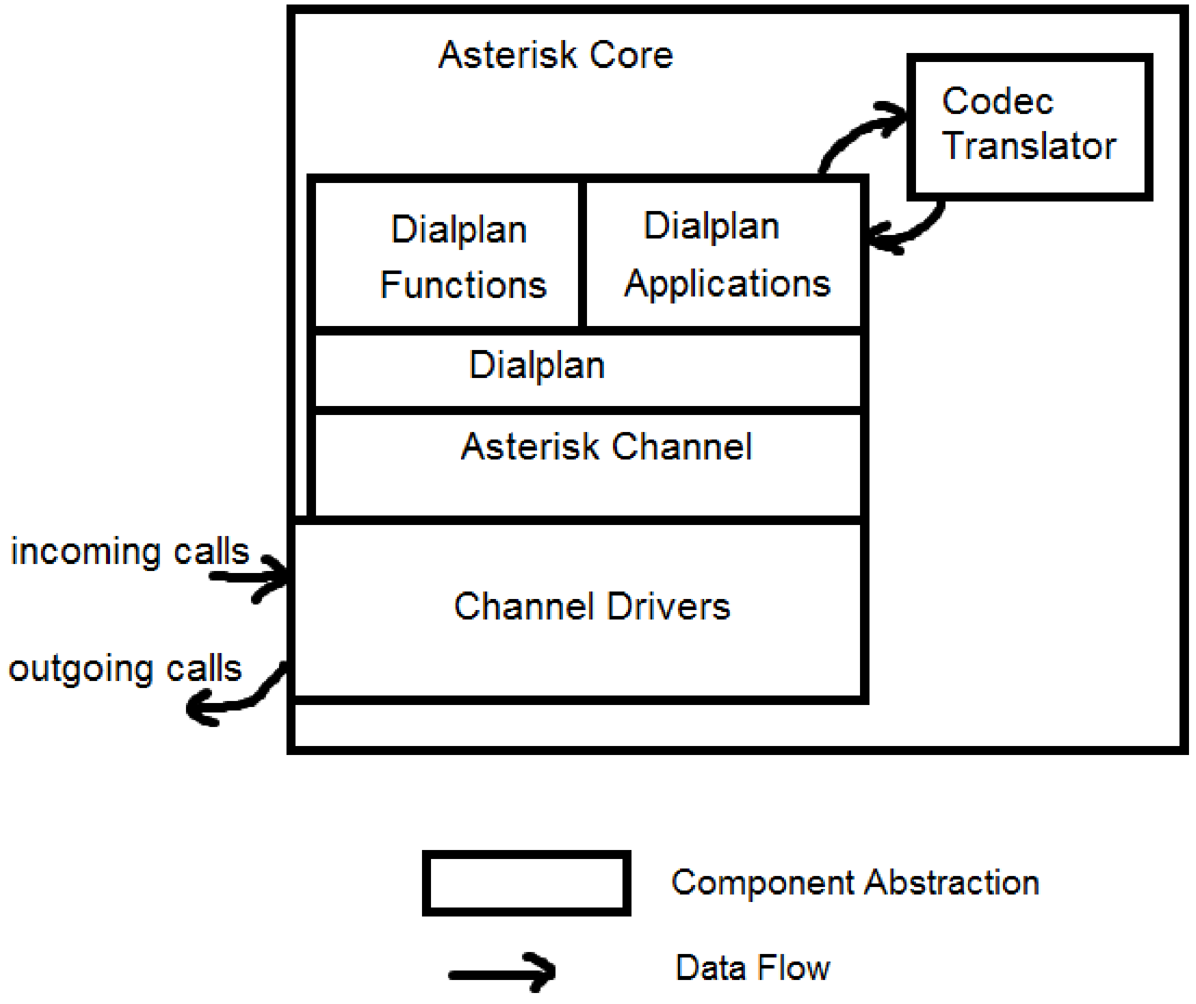
KLOC: ~800
Project start-up: 1999
Number of major releases: 5
Number of developers: 90
Size of user community or number of installations: >2 million downloads
Major stakeholders: The stakeholders in Asterisk are then end users: mostly companies that need a phone system. Digium, the company that maintains Asterisk is a stakeholder in that they make their revenue off of Asterisk related products (add-ons, training, hardware, etc).
Use of concurrency:

Each protocol that Asterisk is configured to use will have a tread that monitors the network for incoming calls. Every accepted incoming call spawns a new thread that runs the dialplan.

Implementation language: C
Supporting software: There are many add-ons to Asterisk, but there does not appear to be any dependencies.

High level architecture

Diagram of software architecture



High level scenarios

Bridged Call

1. Incoming call detected by network monitor thread
2. Channel thread created by network monitor thread
3. Channel thread runs dialplan
4. Based on extension dialed, dialplan calls another phone (additional channel spawned)
5. Codec translators used if necessary

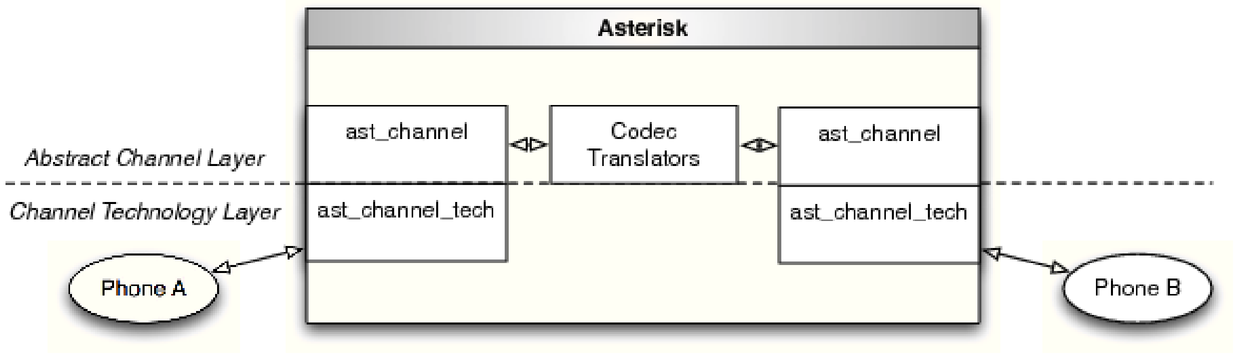


image from chapter 1 of Brown/Wilson book

Data structures or algorithms

Objects are used to abstract away the messiness of real world protocols. The channel driver deals with the outside technology (one channel driver per supported protocol). Inside Asterisk, all phones look the same (ie: they all look like channels). There are also application and codec translator components. Information is passed between these objects as frames, with a different frame type for every media type (audio, video, modem, etc).

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

Frame data structures are used to pass data between components.

Architectural style: Object orientation and custom data structures are the main style of Asterisk's architecture. There are only a few important interfaces: channels, codec translators, applications. The flexibility of Asterisk's comes from the flexibility of the implementations of these interfaces. Once a call is set up, however, the architecture starts to look like a bi-directional pipeline: data enters via the channel driver, is sanitized and passed to the channel, which passes it to a codec translator (if necessary), which passes it to another channel, which gives it to another channel driver, which sends the data out to the other phone.

Major evolutionary changes: The architecture of Asterisk has remained remarkably stable ever since the first release.

Performance bottlenecks: Bandwidth is a major performance bottleneck. This only applies to protocols that go over the web, but these are of ever increasing importance. Especially with video calls becoming more popular. Codec support is also an issue – as of publication, video codecs did not have very good support. Without the proper codecs, performance cannot be optimized.

Real time: Most aspects of Asterisk must be responsive in real time. The Network monitor threads, the channels, the codec translators, the dialplans and applications all need to be responsive enough to seem real time to the human users.

Notation for architecture: The notation used to describe the architecture included several sequence diagrams and block diagrams.

Methodology: There have been five major releases of Asterisk over a period of over 10 years. This indicates to me a spiral model might be close to the truth. The releases are sufficiently far apart that I wouldn't call it agile, and there is clearly a cycle of development, release, and review.

Appendix:
Kruchten’s eight context attributes applied to Brown/Wilson systems

Kruchten attributes described in his slides 17-24 of
https://files.me.com/philippe.kruchten/sbz0ma
Also in: https://files.me.com/philippe.kruchten/1q00nw

1. **Size:** L
2. **Criticality:** Med
3. **Age of system:** L
4. **Rate of change:** Lo
5. **Business model:** open source
6. **Stable architecture:** Lo
7. **Team distribution:** VH
8. **Governance:** NA