Survey of software architecture V0.0

Name of system: Mercurial
Reviewer: David Dietrich
Date: September 17, 2011
Author of software: Matt Mackal
Author of book chapter: Dirkjan Ochtman
Five star rating of book chapter: ****
Purpose of system:

Mercurial is a Distributed Version Control System (DVCS). A Version Control System (VCS) is a system used to track changes to source code by developers. Tracking these changes enables developers to rollback mistakes, check out particular versions, and create specific branches of source code without the headaches associated with manually copying and merging code.

The benefit of Mercurial above other VCSs is that Mercurial is distributed. Instead of having a central repository where developers commit their changes, developers are instead able to commit changes to their own local version of the source code. This has a major benefit that the developer does not need to connect to the base repository to commit changes. Most operations are off-line and do not require an internet connection. This developers changes then can be pulled by other developers and merged into their own source tree. It is also possible to create a central repository where this developer could push their changes. Other developers could then pull from this central location.

Mercurial was designed to be fast and extensible. Because there is no need to contact a server to perform most commands it is high performance when compared to centralized VCSs. There is not only an API for extending Mercurial, but it is also possible to use the reflection properties in Python to re-implement most of the core functions.

Basic metrics

KLOC: I am only counting lines of code in the actual mercurial project, not the extensions that the developers include with the source code download.

Sum of lines in py files: 41
Sum of lines in c files: 2.3
Sum of lines in css files: 1.3
Sum of lines in js files: 0.3
Sum of lines in tmpl (web template) files: 4

Sum of all file types = 48.9

Project start-up: April, 2005 (first release of mercurial (v0.1) = Apr 19)
Number of major releases: 19 (from version 0.1 to 1.9)
Number of developers: ~10

Size of user community or number of installations: The user community of Mercurial is large. There are a number of projects using Mercurial

(http://mercurial.selenic.com/wiki/ProjectsUsingMercurial). As well, a number of sites offer Mercurial hosting (http://mercurial.selenic.com/wiki/MercurialHosting). All Mercurial development discussion is handled through the mailing lists (http://mercurial.selenic.com/wiki/MailingLists). Unfortunately the most recent survey of Mercurial users was done in 2006 and thus the numbers are likely far off their current values.

A quote by Matt Mackall in an interview states that the Mercurial project has “hundreds of contributors and tens of thousands of users”.

A big benefit to Mercurial is excellent Windows support. This increases the potential user community significantly.

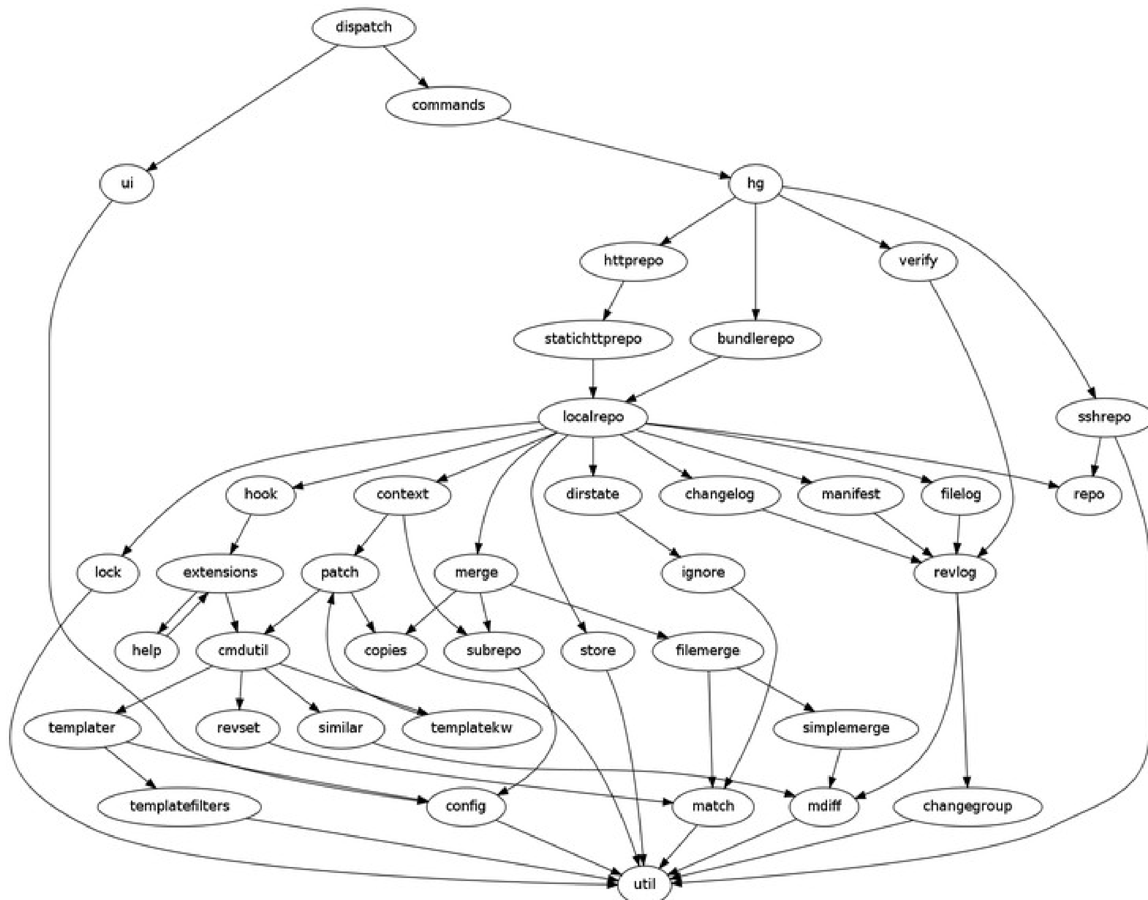
Major stakeholders: Mozilla, Microsoft, NetBeans, OpenOffice, FogCreek, Google Code offers Mercurial (used by several Google projects such as Go).

Sponsors page on Mercurial (http://mercurial.selenic.com/sponsors/) website list's sponsors of Mercurial project.

Use of concurrency: Not used
Implementation language: Python (with C for performance critical parts)
Supporting software: No supporting software

High level architecture

Diagram of software architecture



Each bubble represents a class. Each arrow represents an import. If an arrow goes from class A to class B, then A imports B.

High level scenarios

1. User committing to local repository

- User enters command “hg commit”
- Program enters dispatch
- Creates UI and commands objects
- Commands object recognizes you are committing, enter commit function
- hg creates a localrepo object (don't need a remote repo for this)
- localrepo utilizes context, patch, cmdutil, and util for the commit
- localrepo places information messages in the changelog

2. User pushing to a centralized repository

- User enters command “hg push http://selenic.com/hg”
- Program enters dispatch
- Creates UI and commands objects
- Enter push function in commands class
- Push checks the UI object for destination (http://selenic.com/hg)
- UI object outputs message to user saying it is pushing
- hg object creates a httprepo object
- This utilizes the repo object to get repository information such as branches

Data structures or algorithms:

1. Directed Acyclic Graph : Not just conceptual. Repositories are stored using a tree structure.

2. “Revlogs” : Revlog stored for each file. A revlog is an efficient storage algorithm built around fast access and minimal disk seeks for storing revisions. Layered architecture, each tier being: Changelog (commit information), Manifest (patch information), Filelog (file information).

3. Dirstate : Representation of what is in the working directory at any point. Keeps track of checked out revision, cache of working directory when it was last traversed.

Control flow and/or data key to the architecture if any: Revlog data contains all of the information for a revision and is extremely important.

Architectural style:

1. Peer-to-Peer
   - For operations such as Clone, Push and Pull where you are accessing remote repositories. This can not really be thought of as a Client-Server architecture as there is no distinct server. The repositories are all just clients.

2. Layered architecture of Revlogs (see Data Structures and Algorithms section)

3. Plugin Architecture for extensions
   - Extension locations are placed in a text file that is read by the hg program. Extensions are then able to define a number of data structures and functions that the main hg program will recognize.

4. Object-Oriented Architecture
   - Python classes are used throughout

Major evolutionary changes: The general architecture of Mercurial has remained constant over the years. However there have minor changes over the years. There have been optimizations to existing algorithms, extensions have been merged into the Mercurial core and new functionality has been added.

Performance bottlenecks:

Merging and diffing are the only other place where you can potentially see poor machine performance. These are both implemented in C and are designed to be as fast as possible to counteract this.

Pushing, pulling and cloning remote repositories rely upon network communication so poor bandwidth can be a performance bottleneck. Minimizing the amount of data sent across the network reduces the possibility of this becoming a bottleneck.

In the past committing large files created performance degredation. This was due to large algorithmic complexity in the delta algorithm (delta = a way of sending data by differences, instead of compressing and sending the whole file, ie: how source control works by storing a base file and then differences between versions). This has since been optimized, and mercurial is now able to handle large files with relative ease.

Real time: For commits, delta generation of files is important. This is the most performance critical part of the application.

Notation for architecture: Import diagram (as given above). No other architecture notations are used in the project.

Methodology: Agile.

Source code is always buildable and the project is always in a shippable state. There are no things like on-site clients though.

There is a single “sprint meeting” per release. These are 2 day long physical meetings where many developers get together and hack out some code. This would seem to indicate a Scrum development policy. Each release (ie: 1.3) is a sprint. It does not appear they have Scrum meetings though.

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

1. **Size:** M
2. **Criticality:** Med
3. **Age of system:** M
4. **Rate of change:** Med (optimizations, new functionality, extensions merged into project)
5. **Business model:** Open-Source
6. **Stable architecture:** Med
7. **Team distribution:** Med (there are developer meetings for every release that are in person, but otherwise the team is composed of developers all over the world)
8. **Governance:** Lo