# Goals of Linux Agent

* Platform agnosticism
  + Run on a variety of platforms *(Ubuntu, RHEL, SUSE, Meego, ESXi, Xen, etc.)*
* Desktop, Server, Embedded
* Dynamic and static linking options
* Maintainability
  + Documentation
  + Installation

# Linux Agent components

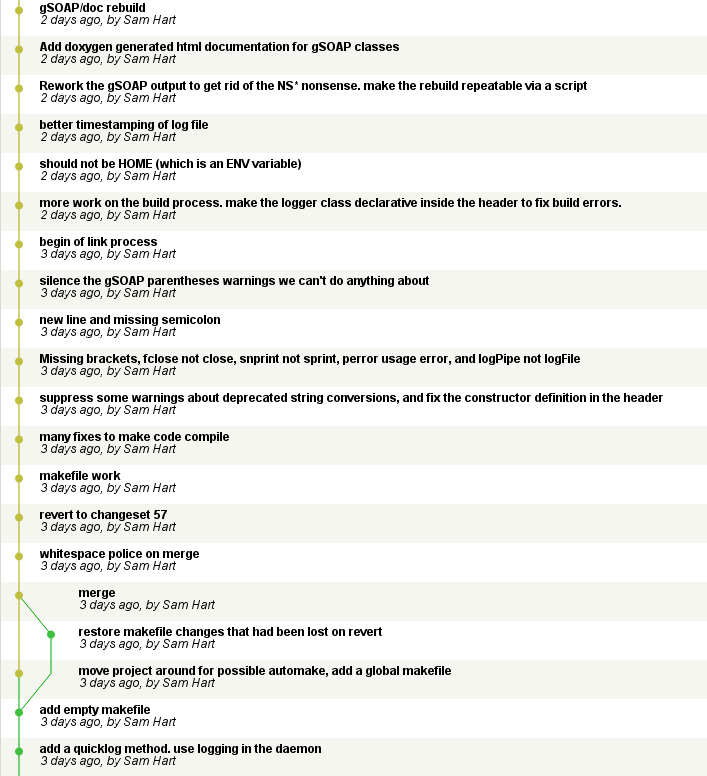
Current plan is to have Linux agent consist of two components:

* A “Steward” which interacts with CCMS, obtaining commands, updating status, etc.
* A “Dispatcher” which the Steward issues commands to, which abstracts the underlying platform specificities.

## “Steward” Linux Agent Design

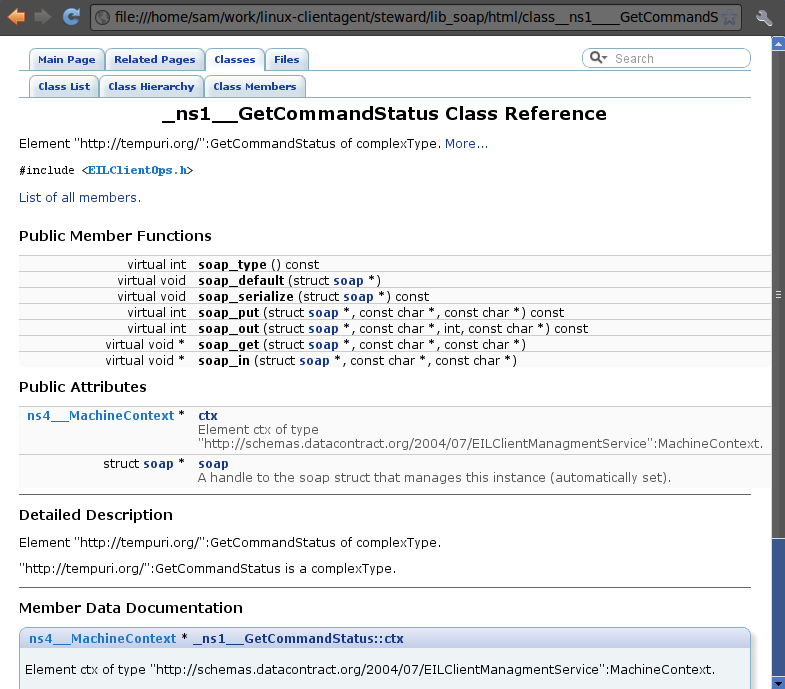
* Previous goal had been to utilize existing C# code in Mono.
  + This lead to complications on platform support re: inhomogeneous Mono versioning across specific Linux distributions and its absence on others.
* Current goal is to utilize lower level gSOAP C/C++ API and write the “Steward” Linux agent in C/C++.
  + The Linux agent will be “thin”, operating only with CCMS and the Dispatcher.
  + Most of the work will be handled by the Dispatcher.
  + gSOAP will allow for maximum compatibility with as many diverse Linux platforms as possible.

## “Dispatcher” Design

* The “Dispatcher” has not changed in design since the decision to switch from Mono to gSOAP.
* Dispatcher will be a central BASH/ASH script (BASH will be present on all Desktop/Server platforms, ASH on the embedded platforms).
  + Dispatcher will include “suite scripts” tailor made for platform specific differences dependent upon distribution.
  + The goal of the Dispatcher is to abstract any platform specific design choices away from the rest of the Linux agent.
  + The Dispatcher will also be responsible for automatically patching/upgrading the Linux agent (“Steward” and “Dispatcher”).

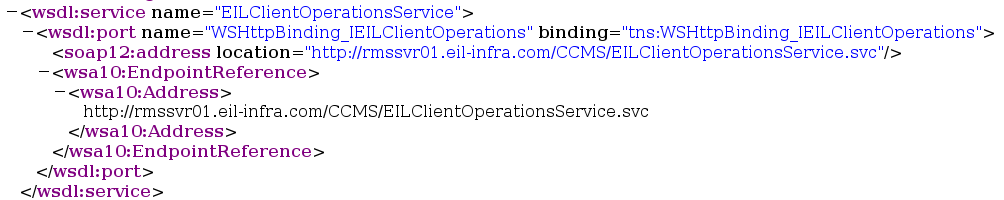
11/15/2010 Update

# What is done

* All current work can be found inside the lab network at a Mercurial repository <http://10.4.0.60/>
  + Mercurial is a Distributed Version Control System, it is used for source control and can also be used for auto-patching on certain clients (Ubuntu, RHEL, SUSE, possibly Meego) in the future. It is cross-platform, open-source, and available for Linux and Windows.
* **Dispatcher**:
  + Framework is mostly complete
    - Installs, upgrades, uninstalls, purges, etc.
  + Scripts are stubbed out
  + Unit testing framework in-place with existing functionality tested
  + Hooks for “Steward” once it is ready
  + Helper tools complete
* **Steward (C/C++ Linux Agent)**:
  + Runs as a secured daemon on Linux
  + gSOAP code has been generated and documented (using Doxygen)
  + System logger in-place, connects with Dispatcher for install/logging information
  + Stubs for gSOAP integration “glue” once it has “firmed”

# Problems/Work remaining

* gSOAP has exposed possible issues in CCMS generated WSDL.
  + WSDL generated contains un-resolvable domains. As an example:



* + - This *may* explain some of the issues we were having getting the C# code working in Mono
    - A work-around in Linux has been to add this un-resolvable domain to /etc/hosts, though I am not certain that is the best long-term solution.
  + The .NET framework masks formatting issues which gSOAP exposes with regard to certain specific command interfaces. This, in and of itself is only a problem in that it complicates the resulting gSOAP API.
    - As an example, “MachineContext”, which is required for each command request from CCMS, winds up being assigned a “Read Only” attribute underneath each command class inside the resulting gSOAP code. E.g.,:  
      \_ns1\_\_GetCommandToExecute->ctx *(Read Only)*
  + These can be overcome, but additional initial work must be done to figure out *why* it is happening and automate the repair of it in the future.
* Presently, the “Steward” can ping and query the possible interactions with CCMS. The work that remains is to resolve the aforementioned MachineContext issue so it can actually receive commands.
* Additional “glue” between the “Steward” and the “Dispatcher” once commands are being received must be completed.
* Additional “Dispatcher” suite scripts (currently only hard requirement is “Reboot”, surely there has to be more than just that?)
* Additional unit tests to be determined.

11/29/2010 Update

# Current Progress:

* Stock gSOAP was not generating *any* XML header information.
  + Work had to be done to determine what XML header details were needed by CCMS- what it was expecting. This involved a bit of reverse engineering (Wireshark, custom gSOAP tools, etc.)
  + CCMS was expecting WS-Addressing construct headers.
* Stock gSOAP present in distributions had spotty support for WS-Addressing (WS-A), which is needed for the Linux Client Agent.
  + Options were to either recreate WS-A headers by hand, or rebuild unified gSOAP base.
  + Unified the Linux Client Agent under gSOAP 2.8.0 from upstream.
  + Rebuild gSOAP packages (with WS-A support) for Ubuntu, RHEL/CentOS. Plans to rebuild for SLED/SLES.
* By upgrading to gSOAP 2.8.0 and including WS-A support, upon gSOAP binding rebuild the “inside-out” API issues from last update were partially resolved (at least for header generation).
  + This also cleaned up the resulting API documentation greatly.
* Currently integrating WS-A headers into the steward application.
* Currently replacing stock gSOAP service wrapper with custom wrapper:[1]
  + gSOAP’s wsdl2h and gsoapcpp2 tools generate a service proxy wrapper which, thus far, the Linux Client agent has depended upon.
  + This service wrapper is too inflexible with regard to XML header generation and must be replaced.
  + Work has begun to replace it with custom, in-house service wrapper (stewardService.cpp)

# Problems/Work remaining:

* Networking:
  + At present, none of the Linux development environments have complete network access.
    - LENM58P-Ubuntu01, LENM58P-RHEL01, and LENM58P-SLED01 cannot obtain IP addresses.
    - UbuntuDev (VM) can obtain an IP address, but seems to have incorrect routing.
  + ***Update: 11am*** *– Networking issue resolved, Linux machines back online.*
* Namespace:
  + There is a possibility that the automated gSOAP generated namespaces will need to be regenerated.
  + In reviewing the W3C submission on WS-A, there seems to be a more strict naming convention for WS-A then gSOAP seems to be taking.
  + Until I can start testing the gSOAP application in the lab, I will not be able to tell whether this needs to be regenerated.
    - ***Update: 11am*** *– After testing on the network, have discovered that namespace collision is not the only problem, we must also use subsets of the headers dependent upon which CCMS command we are executing. See note [1] above.*
* “Steward” service wrapper stubs fleshed out once current WS-A headers are finalized for other commands.
  + This will involve additional reverse engineering (Wireshark) of CCMS commands, unless there is lower level CCMS design documentation (protocol level, operations taking place behind the current C# code).
* Additional “glue” between the “Steward” and the “Dispatcher” once commands are being received must be completed.
* Additional “Dispatcher” suite scripts
* Additional unit tests to be determined.
* Upgrade RHEL, SLES, Ubuntu? (Recent releases of each)
  + Recent Ubuntu was not LTS
  + Current RHEL is quite old, highly recommend upgrade it at least
* Would a web-based project tracker be helpful in others tracking progress and/or interacting with the progress?
  + For example, trac or Redmine integrate well with existing tools.