THE BORG “ARCHITECTURE” AS A METAPHOR FOR FLOSS

Andrea Trentini
DICo–UniMi, Via Comelico, 39, 20135 MILANO, ITALY
andrea.trentini@unimi.it

ABSTRACT
The Borg (yes, the ones from the Star Trek series) are an alien species with a unique peculiarity: every “individual” is connected to any other through a high speed communication network, thus any knowledge acquired by any member of the community is instantly spread/shared and taken advantage of. This mechanism makes The Borg very knowledgeable and adaptive, they are virtually invincible in battle and wherever they go they assimilate other cultures.

This article argues that The Borg architecture and capability is a close metaphor of FLOSS (Free Libre Open Source Software) paradigm... with some differences of course, luckily. FLOSS has some limitations but also some advantages over the fictional Star Trek alien civilization. First of all, FLOSS is not Evil.

Authors will use, as an example, a system (codenamed NetLAMPS) they recently built (based entirely on FLOSS) to gather road/traffic status information. This software is enough complex (involves many technologies: mail, cell phones, gps, webcams, network, etc.) to represent a reasonable example and at the same time enough manageable (less than 3k LOC - Lines of Code) to be almost entirely explained in this article.

KEYWORDS
Software Engineering, Free Software, Integration

1. INTRODUCTION

First of all we will introduce the reader to the context and briefly explain the concepts used throughout this article. We are dealing with: 1) a fictional alien species, The Borg (http://en.wikipedia.org/wiki/Borg_(Star_Trek)); 2) FLOSS (http://en.wikipedia.org/wiki/FLOSS) software and licenses; 3) a software project aimed at road quality monitoring (to be used as an example).

This article will try to demonstrate the parallels between the fictional alien species and the way software is built using a pure FLOSS approach. Why should we need such a metaphor? We need it to fully understand the power of the FLOSS approach, a power that immeasurably improves software evolution. Since every Star Trek fan knows (and fears) the power of The Borg, it should be easy to explain and understand (with this metaphor) the power of FLOSS. For the sake of avoiding terror let us say that the metaphor only includes good aspects, nobody should fear FLOSS software.

The Borg (yes, the ones from the Star Trek series) are an alien species with a unique peculiarity: every “individual” is connected to any other through a high speed communication network, thus any knowledge acquired by any member of the community is instantly spread/shared and taken advantage of.

This mechanism makes The Borg very knowledgeable and adaptive, they are virtually invincible in battle and wherever they go they assimilate other cultures. “Resistance is futile” is their motto.

“Knowledge is power” is a worldwide acknowledged principle. Humans learn from own experience and errors (they can share knowledge of course, but the process is not instantaneous), The Borg learn instantly from everybody's experience and errors. It's also important to say that The Borg are not only able to learn many things, they can also implement/apply their knowledge very powerfully since their bodies are a mix of live parts and artificial parts. These artificial organs (built using technologies assimilated from other species) represent the Borg way to evolution. They ameliorate themselves at a faster pace than natural evolution: they don't need to follow the natural “survival of the fittest - genetic sieve” (Darwin, 1883). In very short terms: they know and they can, fast.
Free Libre Opensource Software (FLOSS) is a general name to indicate the set of software packages distributed with liberal licenses (such as the GNU GPL - http://www.gnu.org/licenses/gpl.html) that grants any user the rights to: 1) run the software, 2) study the source code, 3) modify the code, 4) include the software in a larger project, 5) re-distribute the (un)modified software. Usually, but it's not always the case, the only limitation to exercise these rights - actually applicable to the modification and redistribution ones - is that the redistribution must be acknowledged (original authors must be identifiable) and that the license must remain the same. This is true for copyleft (http://en.wikipedia.org/wiki/Copyleft) licenses, while it's a bit different with non-copyleft (http://en.wikipedia.org/wiki/Permissive_free_software_licence) ones.

NetLAMPS.org Italy is a beautiful country, we know. And we agree with you if you are referring only to “natural” beauty. If you look at public services and public infrastructures you will probably get a different picture. Among the public infrastructures, roads and signaling are a mess and our government does not invest much in maintenance. Holes, bad asphalt, no (or badly maintained) signals, stupid traffic laws (such as the circulation of odd/even plates in alternate days), the use of speed-cameras to raise money (many small town budgets are actually based on speed tickets collected from vehicles passing by), etc. We will not delve into the socio-economical-political issues that brought us were we are(n’t), please just take the awful situation for granted. In particular, the author is the founder of a group of civic motorbikers trying to monitor/lobby our public institutions (from nation-wide to local-wide). The group aims at raising the quality of roads and traffic management, with a special regard to motorbikers. Since a motorbike is far more subject to general road defects (e.g., an asphalt irregularity may cause a two wheeled vehicle to trip over) than a car, we are more concerned than car owners about the quality of road maintenance. To keep track of road defects, irregularities, works, etc. we decided to build a software system to support a community of concerned motorbikers. Such a system should be able to gather geo-localized information (mainly photos) from users, categorize them, display this information on maps and create downloadable files (for GPS navigators). So we built NetLAMPS.org. Still a work in progress, but it works. At the moment, since a large community (critical mass) is yet to come, the author and some colleague test the system by simulating many users: traveling around getting photos and “asphalt-o-grams” (see below) and then uploading everything on the website. The architecture is simple:

- **Input:** geolocalized items (mainly photos), directly uploaded or sent by email/sm/sms
- **Output:** webpages, maps, downloadable files for gpases

The generic “geolocalized items” are mainly photos, but the essential aspect is that every item must contain at least the following information: **who** created the item, **what** this item represents (a hole, bad signal, speed-camera, etc.), **when** this item was created, **where** this item is located (address or lat+long pair).

Items are tagged with the above information by using a “Comma Separated Values” format in the name of the object. We have three types of informational items: geolocalized plain text, geolocalized photo, geolocalized “asphalt-o-gram” (see Figure 1). An “asphalt-o-gram” is like a cardiogram or a seismogram, i.e., a graphical representation of an oscillating signal. In our case the oscillating signal, measuring the smoothness/roughness of the asphalt, is generated by an accelerometer applied to the testing vehicle. If the asphalt is smooth the graph is flat, if not you see sharp oscillations. The capture hardware/software we built (see below for the details about how we did it) generates grams by capturing few seconds of signals per item. All the gathered items are then displayed on maps, we use googlemaps.
2. DESCRIPTION OF SYSTEM COMPONENTS AND APPROACH USED TO INTEGRATE THEM

This section describes the reasoning and methodology used to satisfy every need. Each subsection describes a small subsystem. The focus is on the procedure followed to find and integrate existing software into the system being built. When searching for a solution we usually follow a “procedure”, albeit not formal. First we try to look at something already installed on the system, then, if not satisfied, we broaden the search field.

The order of search sources, from narrow to broad, is the following:

1. manpages (i.e., already installed software) with “man -k” command
2. check debian/ubuntu packages (i.e., not yet installed) with “apt-cache search” (http://en.wikipedia.org/wiki/Advanced_Packaging_Tool) command
3. sourceforge.net website et similia (i.e., not yet packaged into mainstream GNU/Linux distributions)
4. google/wikipedia (maybe not yet developed, just ideas published on the web)

In the following subsections, when stated, the “time to assimilate” is the approx time, in man-hours or man-days, to choose the product, to learn it (not completely of course, just the minimum to integrate it into the system) and to integrate it (usually by writing some glue code). Remember also that every piece of software we used comes for free, can be studied to see if it fits, can be modified and redistributed. And we took advantage of every aspect.

Some statistics about the glueing software, mostly shell scripts:
- Number of scripts=46
- Number of bash scripts=45
- Number of python scripts=1
- LOC average=60
- LOC min=21
- LOC max=210
- LOC total=2778

LOC numbers include comments (variable) and license header (17 lines per script) and config files are not taken into account even if large (e.g., motion.conf is 620 lines and must be edited with particular care).

Note on package availability: when not explicitly referenced, they are taken from the Ubuntu repository (http://packages.ubuntu.com/search?keywords=<PACKAGE NAME>) instead of their own author's website.

#Website
Of course, after registering the domain, we needed a CMS (Content Management System - http://en.wikipedia.org/wiki/Content_management_system) or something similar to manage all the web contents, editing raw HTML is not an option any longer. Since we’re already familiar with wikis (http://en.wikipedia.org/wiki/Wiki) we decided to use some kind of wiki software. How to choose the right one? Go to one of the matrices, web sites that compare various software, such as Wikimatrix (http://www.wikimatrix.org), insert your requirements (e.g., flatfile database, plugins, themes, preferred language, etc.) et voilà, the choice is almost made. We just had to decide among the four or five CMS with the required features. The resulting choice was http://dokuwiki.org, also because we needed a specific plugin (http://www.dokuwiki.org/plugin:googlemaps) capable of displaying googlemaps from KML (http://en.wikipedia.org/wiki/KML) files. Time to assimilate: half a day (install and set-up).

#Geotranslation
The need to translate a street address into a pair of coordinates - latitude and longitude - was satisfied using publicly available services, We wrote scripts for Google and Yahoo. Both sites have mapping and translation capabilities, they are targeted at human usage, but it's very easy to extract information in an automated manner. We only needed a scriptable tool to get web pages, the king of these tools is “GNU wget” (http://www.gnu.org/software/wget), coupled with some classical Unix filters. We wrote a small shell program that takes a complete textual address and gives back the LAT+LONG pair, if found. Time to assimilate: couple of hours (studying the original web page to create an automatic script + writing the actual script).

#Getting photos
In our system we can receive geolocalized photos from users, but we also created a tool that can gather images automatically while we drive around. It's a combination
(see Figure 2) of a netpc, a gps, one or more webcams and a keypad or a Nintendo Wiimote (why a Wiimote? See subsection “asphalt-o-grams”) to mark spots. Current position is marked through the GPS, webcams get images periodically or when they detect movement, the numeric pad or the Wiimote is used by the driver to mark a hotspot when going around (e.g. he presses “1” for a hole, “2” for a bad road sign, etc.). This “gatherer” seems complicated to build... well, it's not. Actually the hardest step was finding a webcam completely compatible with GNU/Linux and not too sensitive to exterior lighting... alas most webcams fit well inside, but when placed outside they get saturated and they give almost white images. Still images and movies are managed by the “motion” package, a well known software under GNU/Linux, very configurable and robust, it generates timestamped images and SWF movies. At the end, back at home, we have to cross reference all the data fluxes (photos, positions, markers) to generate geolocalized marked photos to be uploaded on the website. We wrote a couple of shell scripts that do this integration, they are mostly based on traditional Unix filters (cut, grep, etc.). Final images are labeled with a copyright text using the “imagemagick” package. **Time to assimilate**: one day for a working prototype, a week to have something robust, also hardware-wise, to be mounted on a motorbike.

**#GPS info.** To gather GPS info we needed a GPS device (we opted for a bluetooth one) and some software to connect and get data. Luckily, GPS devices are well integrated into GNU/Linux, the search was very fast, we installed the standard bluetooth stack and the “gpsd” (http://gpsd.berlios.de) daemon and utilities. Then we just capture the logs to track the position in time. **Time to assimilate**: couple of hours (setup an config).

**#Asphalt-o-grams.** Our system can gather geolocalized photos (e.g. from users) of pinpointed road defects, but we are also interested in “spread” defects, such as bad paving which is not localized in a single spot. In this cases a single photo has little significance. So we decided to measure vibrations generated from the road-vehicle interaction. At first we searched for some kind of industrial accelerometer to be mounted on the motorbike... then we had an insight: why don't we use a Wiimote (http://en.wikipedia.org/wiki/Wii_Remote)? Is it possible? Can it be connected to a standard PC? The answer is “yes”, after searching a little bit we found a python script (http://abstrakraft.org/cwiid/browser/trunk/wmdemo/wmdemo.py) to get data from the Wiimote accelerometer (via bluetooth). We modified it to make it a simple logger and that was it. Next we needed a tool to create graphics from the timestamped data... again, no need to search long, “R” (http://www.r-project.org) is the answer, the script we wrote is 10 lines long. The result is depicted in Figure 1, we generate one graph every 50-100 lines of accelerometer data, since it's a moving accelerometer we cannot take all the data in one shot, otherwise we could not associate it to a single position. **Time to assimilate**: one day (searching for info, find python script, modify it, test, tune, create R script).

**#Phone management.** Since we want to receive info from users on the road, we decided to create a tool to receive SMS (Short Message Service) and MMS (Multimedia Message Service). And we don't require our users to have a GPS with them, so we let them specify just an address that we will translate to lat+long on our system (see subsection “Geotranslation”). Phone management under GNU/Linux is very stable, a couple of packages is all you need: “gammu” (sms/mms management) and “obex” (phone filesystem management). Humorous remark... the only real problem we met here was that the phone we used (Nokia 2600 Classic) to receive messages has a severe design fault: the data cable and the power cable CANNOT be plugged at the same time due to physical dimension of the connectors. We had to rasp both connectors to fit them! System must stay up 24/7 unattended, we don't want to be there periodically to charge the phone. **Time to assimilate**: half a day (just to tune the config).

**#File management miscellanea.** Throughout the system we need tools to create files in various formats (KML, CSV, etc.) and to keep everything in sync... Again, no need to write anything from scratch, just use these packages: “xml2” (pack/unpack XML files), “tidy” (clean/validate HTML), “parsecwiki” (generate documentation), “mirrordir” (sync entire directories sending only the differences). **Time to assimilate**: minutes (to read man pages).

**#Mail handling.** The scripts to handle incoming mail (remember that users can send geotagged photos by email to a virtual email address) are mainly based on the packages: “formail”, “munpack”, “uudecode”. **Time to assimilate**: less than an hour (writing a small script).

**#Clock handling.** To keep clock synced with timeservers on the net we just installed the “ntpd” daemon. **Time to assimilate**: minutes (install and config).

**#Miscellanea.** To create a textual interface that launches all the various scripts, based on user choice we used an ncurses interface called “dialog” (http://invisible-island.net/dialog/dialog.html). **Time to assimilate**: minutes (read a man page).
3. CONCLUSION AND OPEN ISSUES

A “FLOSS vs. The Borg” comparison. Notable differences: 1) assimilation is voluntary, not forced; 2) assimilation is a social and cultural process, not a physical one; 3) FLOSS is meant to liberate the assimilated, The Borg to take control of them; 4) there’s only one species superior to The Borg, called Species 8472 (http://en.wikipedia.org/wiki/Species_8472), but we don’t see any metaphor of Species 8472 in our world that could defeat FLOSS. Notable similitudes: 1) FLOSS helps us all to connect to the “collective consciousness”; 2) FLOSS resistance is also futile (at least we hope). Capital difference: there is not a FLOSS Queen (female or male) neither a “Royal Protocol”, FLOSS is based on strong legal tools such as GNU GPL license or BSD license that ensure collective control of free software we use, no “central point of control” is possible, although one relevant evil legal tool constantly threaten FLOSS: software patents (http://endsofpatents.org).

The publication of a piece of software under a FLOSS license lets every other developer in this world use that software in his work. No bureaucracy needed, no royalties, no limitations: the moment you see it is also the moment you can use it. Only technical limitations of course: understanding the code, compatibility of language/architecture, bugs, etc. But of course these limitations apply also to proprietary software. Provided that you distribute your product with the same FLOSS license (not even always required) of the software you use/integrate, you can pick anything from the:

- >25000 packages of the Debian (http://debian.org) repository
- >230000 projects of the sourceforge (http://sf.net) website
- who-knows-exactly-how-many pieces of FLOSS-licensed code on the net

This article shows a real example of fast and economic software assimilation, only possible in the FLOSS universe, where users have many rights and almost no barriers. The same project, developed under “normal” circumstances, would have been either: entirely developed from scratch (note that the overall LOC figure for all the packages integrated in our project ranges way over one million, while we developed glue scripts for a total of less than 3k LOC), with a Time To Market larger than ours; partly licensed from proprietary vendors at a high price while we spent 0 EUR for software licenses.

There are issues, but they relate to any kind of software, i.e. we need: 1) more effective ways to know about the existence of a program that solves a problem, 2) more effective ways to (pre)evaluate any program. The first can be somewhat “proceduralized” by searching well known sources (as we depicted in section 2: man, apt, sourceforge, etc.) in an orderly fashion. The second one cannot be addressed formally because of the demonstrated impossibility (Turing, 1936) to say that a program can match another one or a formal specification. Of course it can be addressed for example by reading documentation, by digging into forums to know what other use think about one particular project, etc. The two aspects together are a major concern in the FLOSS world than in the proprietary one because of the great availability of software (around 25k packages only in the Debian system), i.e., great choice is time consuming. But it's not that bad: the more you use FLOSS the more you know who to trust, you make your own list of trusted sources of information and software repositories and you learn how to check/weigh a software package very rapidly. And in the plethora of available packages anyone can be pretty sure to find the (almost) right software he needs: at worst some customization will be needed. And remember: FLOSS is not evil, since anybody can take advantage of this paradigm. You can be a “FLOSS Borg” too!

ACKNOWLEDGEMENTS

The author wish to thank Giovanni Biscuolo (g@xelera.eu) for the useful collaboration.

REFERENCES

Darwin, C., 1883, On the Origin of the Species by Means of Natural Selection, D. APPLETON AND COMPANY, UK