GSoC 2010 in Retrospect

P. Dylan McCall (dylanmccall@ubuntu.com)

February 10, 2011

This is a retrospective of my work done with Harvest for Google Summer of Code 2010. It describes the process, some of the approaches taken and lessons learned. I received feedback and assistance from Daniel Holbach and James Wetsby through code review, weekly discussions and casual chit-chat.

I work on Ubuntu[1], a free and open GNU/Linux based operating system. The Ubuntu project participates in Google Summer of Code[2], a program where students, through Google, contribute to open source projects and are paid a stipend for their efforts. I chose to work on Harvest, a new web application for Ubuntu developers, and Daniel Holbach offered to mentor me.

Ubuntu ships thousands of software packages that are developed and maintained externally. There are also other operating systems, like Fedora, that make their own changes to the same software. Harvest aggregates information from source code repositories and bug trackers so Ubuntu's developers can quickly find items they may be interested in. For Harvest, these are referred to as *opportunities*.

Harvest is a young project, and one thing it lacked was a powerful front-end. My project intended to fix that. Meeting with developers and community members at the Ubuntu Developer Summit in May 2010, we created a mockup and set three distinct goals:

- Investigate the application's performance issues as well as short-term and long-term remedies.
- Develop a user interface that helpfully presents the site's content.
- Establish an interactive design that complements the user interface.

Harvest may be a utility for developers, but I considered it important to reflect Ubuntu's commitment to good design and accessibility.

[1] ubuntu.com
[2] code.google.com/soc

Databases and performance profiling

When I began my work, the meat of Harvest's development had focused on data collection. The component to present that data to an end user had not seen as much attention. In its rough state, it exposed some serious performance issues.

Harvest's server code uses the Django web framework, which neatly abstracts SQL databases behind Python objects. That makes it very easy to experiment and develop features, but it also makes it tricky to understand when the database code is running slowly. I set up two profiling tools to help with that: the *Django Debug Toolbar* by Rob Hudson, and django-*profiler* by Dima Dogadaylo.

With those tools I could study the raw SQL queries to identify redundant and unnecessary operations. Daniel and I used that information to fine-tune Harvest's database model, employing indexes and other optimisation techniques suggested in the Django documentation[3].

With our work streamlining Harvest's database code, the most significant performance issue tied neatly to the next stage of the project: with each request, it gathered and returned much more data than the user would want at any given time.

We needed a system that would know exactly what to show the user so we could minimise wasted database hits. We did not want irrelevant information to get in the way or to complicate our database code. In that same vein, we wanted a predictable user interface. When the user searches for something, nothing should be hidden for a technical reason; the system itself should disappear. That desire fed back to our need to enhance the database code, from which a happy medium eventually emerged.

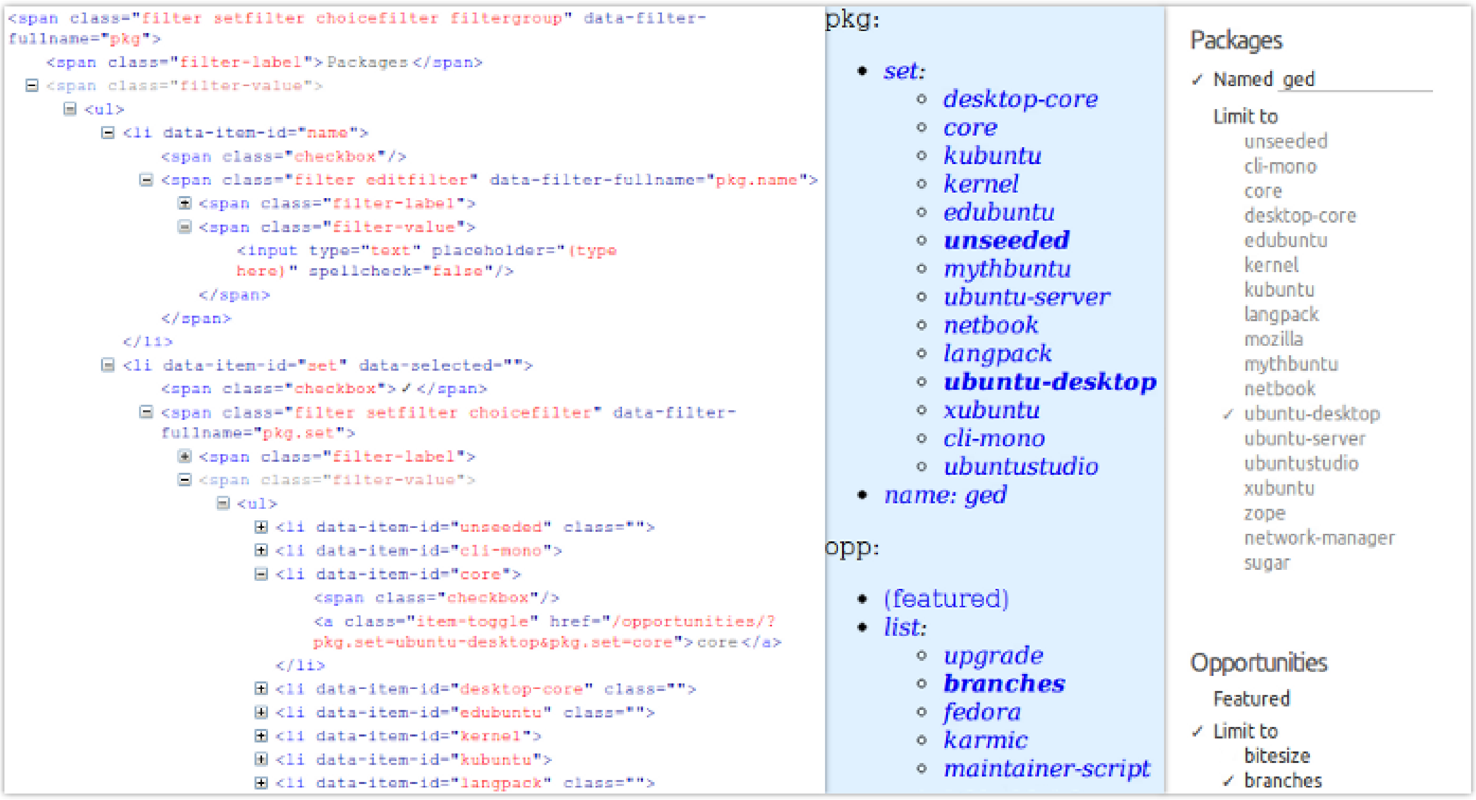
[3]docs.djangoproject.com/en/dev/topics/db/optimization

Navigation and user interface

We decided *Filters* would be a good approach to navigate Harvest's database. The user selects some properties and matching objects are shown as results. Results are presented as a list of packages, which can be expanded to show filtered opportunities. Technically, each filter ties to a specific queryset[4] which the system resolves to a small number of database operations.

Django's modular design cleanly separates a website's logic from its presentation. The Filters feature was an opportunity to explore that technique. Using the built in Forms module as a reference, I made sure the Filters system itself didn't depend on a specific implementation. I kept the majority of its code in a loosely coupled, abstract module that could be used in other projects very easily. The approach proved very flexible and has encouraged tidy, maintainable code within Harvest itself.

During our planning meeting, I was pointed to the idea of a stateless web service. I worked towards that by making the system controlled through the HTTP query string. In this way I minimised the number of session-specific page elements, ensuring that each page returned by the server maps directly to the requested URL. The design means Harvest behaves like a traditional website, improving semantics and browser support. For example, people can bookmark and return to links that show particular selections.



```
<span class="filter setfilter choicefilter filtergroup" data-filter-
fullname="pkg">
    <span class="filter-label">Packages</span>
  <span class="filter-value">
      <ul>
        <li data-item-id="name">
            <span class="checkbox"/>
          <span class="filter editfilter" data-filter-fullname="pkg.name">
            <span class="filter-label">
            <span class="filter-value">
                <input type="text" placeholder="(type
                here)" spellcheck="false"/>
              </span>
            </span>
          </li>
        <li data-item-id="set" data-selected="">
            <span class="checkbox">✓</span>
          <span class="filter setfilter choicefilter" data-filter-
            fullname="pkg.set">
            <span class="filter-label">
            <span class="filter-value">
              <ul>
                <li data-item-id="unseeded" class="">
                <li data-item-id="cli-mono">
                <li data-item-id="core">
                    <span class="checkbox"/>
                    <a class="item-toggle" href="/opportunities/?
                    pkg.set=ubuntu-desktop&pkg.set=core">core</a>
                  </li>
                <li data-item-id="desktop-core" class="">
                <li data-item-id="edubuntu" class="">
                <li data-item-id="kernel">
                <li data-item-id="kubuntu">
                <li data-item-id="langpack" class="">
```

pkg:

- *set:*
  - *desktop-core*
  - *core*
  - *kubuntu*
  - *kernel*
  - *edubuntu*
  - ***unseeded***
  - *mythbuntu*
  - *ubuntu-server*
  - *netbook*
  - *langpack*
  - ***ubuntu-desktop***
  - *xubuntu*
  - *cli-mono*
  - *ubuntustudio*
- *name: ged*

opp:

- (featured)
- *list:*
  - *upgrade*
  - ***branches***
  - *fedora*
  - *karmic*
  - *maintainer-script*

Packages

✓ Named ged

Limit to
- unseeded
- cli-mono
- core
- desktop-core
- edubuntu
- kernel
- kubuntu
- langpack
- mozilla
- mythbuntu
- netbook
- ✓ ubuntu-desktop
- ubuntu-server
- ubuntustudio
- xubuntu
- zope
- network-manager
- sugar

Opportunities

Featured

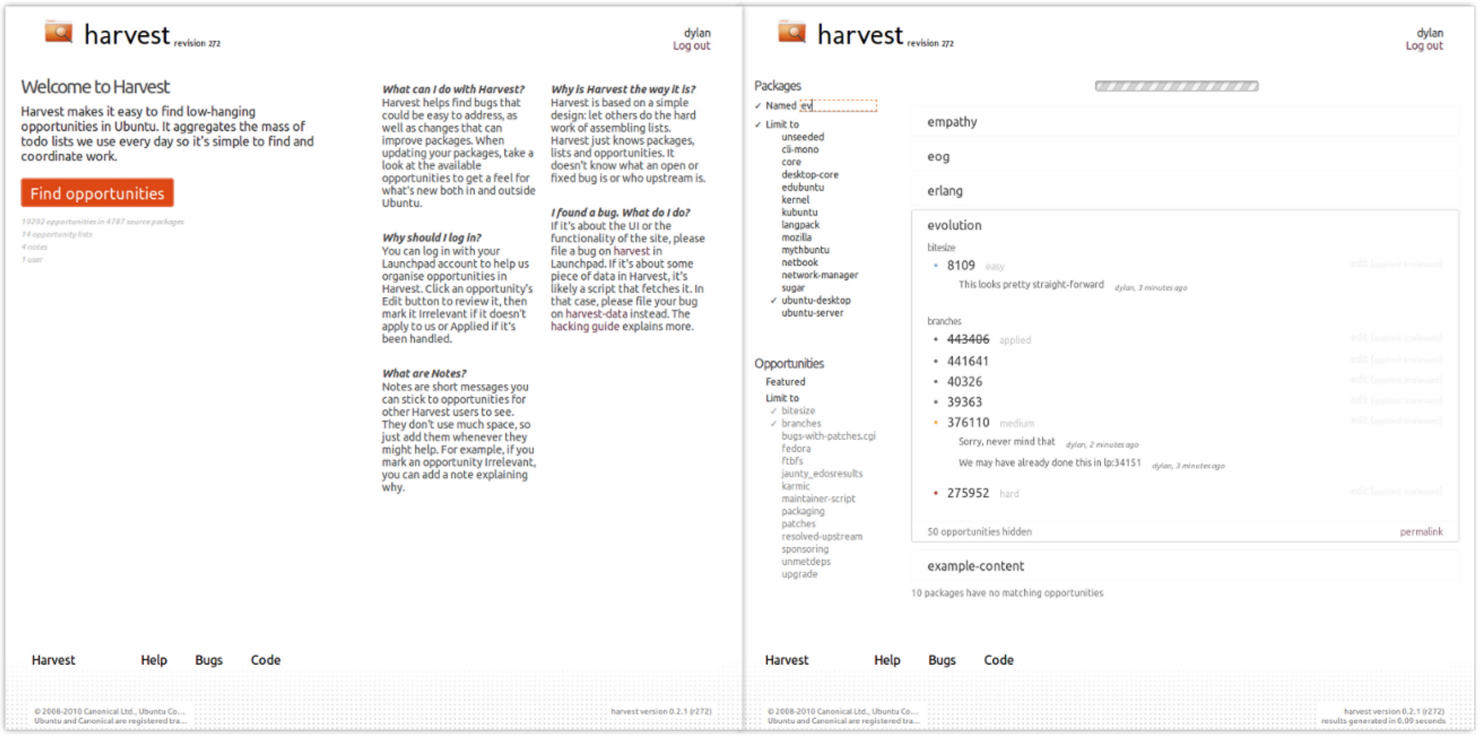
✓ Limit to
- bitesize
- ✓ branches

[4] docs.djangoproject.com/en/dev/ref/models/querysets

Interaction, fit and polish

As Harvest grew in complexity, it became clear that speedy database code was not enough for the user experience we wanted to achieve. The system simply handles a great deal of data, and it would be impossible to avoid the occasional lengthy operation. Through some careful design we could reduce the number of database hits for each action by the user, only probing details for things the user has shown an interest in. The last step in my project was to improve the portion of Harvest that runs in the client's web browser, improving *perceived* performance and making the system more approachable.

I began with a fresh visual design. It evokes simplicity, borrowing elements from Ubuntu's design guidelines[5] to connect gracefully with related sites. Colour is used sparingly so it draws attention to specific details and carries a particular meaning whenever it is applied. Deeper colours have a stronger presence while shading and whitespace is used to organically separate elements.



harvest revision 272

dylan
Log out

Welcome to Harvest

Harvest makes it easy to find low-hanging opportunities in Ubuntu. It aggregates the mass of todo lists we use every day so it's simple to find and coordinate work.

Find opportunities

19292 opportunities in 4787 source packages
14 opportunity lists
4 notes
1 user

What can I do with Harvest?
Harvest helps find bugs that could be easy to address, as well as changes that can improve packages. When updating your packages, take a look at the available opportunities to get a feel for what's new both in and outside Ubuntu.

Why should I log in?
You can log in with your Launchpad account to help us organise opportunities in Harvest. Click an opportunity's Edit button to review it, then mark it Irrelevant if it doesn't apply to us or Applied if it's been handled.

What are Notes?
Notes are short messages you can stick to opportunities for other Harvest users to see. They don't use much space, so just add them whenever they might help. For example, if you mark an opportunity Irrelevant, you can add a note explaining why.

Why is Harvest the way it is?
Harvest is based on a simple design: let others do the hard work of assembling lists. Harvest just knows packages, lists and opportunities. It doesn't know what an open or fixed bug is or who upstream is.

I found a bug. What do I do?
If it's about the UI or the functionality of the site, please file a bug on harvest in Launchpad. If it's about some piece of data in Harvest, it's likely a script that fetches it. In that case, please file your bug on harvest-data instead. The hacking guide explains more.

Harvest Help Bugs Code

© 2008-2010 Canonical Ltd., Ubuntu Co...
Ubuntu and Canonical are registered tra...
harvest version 0.2.1 (r272)

harvest revision 272

dylan
Log out

Packages
✓ Named ev
✓ Limit to
unseeded
cli-mono
core
desktop-core
edubuntu
kernel
kubuntu
langpack
mozilla
mythbuntu
netbook
network-manager
sugar
✓ ubuntu-desktop
ubuntu-server

Opportunities
Featured
Limit to
✓ bitesize
✓ branches
bugs-with-patches.cgi
fedora
ftbfs
jaunty_edosresults
karmic
maintainer-script
packaging
patches
resolved-upstream
sponsoring
unmetdeps
upgrade

empathy

eog

erlang

evolution

bitesize
- 8109 easy
  This looks pretty straight-forward *dylan, 3 minutes ago*

branches
- ~~443406~~ applied
- 441641
- 40326
- 39363
- 376110 medium
  Sorry, never mind that *dylan, 2 minutes ago*
  We may have already done this in lp:34151 *dylan, 3 minutes ago*
- 275952 hard

50 opportunities hidden
permalink

example-content

10 packages have no matching opportunities

Harvest Help Bugs Code

© 2008-2010 Canonical Ltd., Ubuntu Co...
Ubuntu and Canonical are registered tra...
harvest version 0.2.1 (r272)
results generated in 0.09 seconds

With Harvest's visual style reasonably established, I went ahead with the interactive part of that design. Further in the interest of simplicity, I wanted to try a modeless system. Alan Kay writes about the approach in his essay *User Interface: A Personal View*[6]*:* No operation should block another operation; the user

[5]design.canonical.com/the-toolkit

[6]A.C. Kay, "User Interface: A Personal View," *The Art of Human-Computer Interface Design*, B. Laurel ed. Addison-Wesley, 1990, pp. 191–207.

should be able to interact with any control at any time and see an immediate response. This gives some extra room for the interface to be helpful, but it also needs fine-tuning. For example, Harvest enables a filter when the user has shown an interest by editing its value, but only when the change is a positive one such as adding a selection.

Conventional HTML pages are modal because the jobs of generating and displaying pages are decidedly split between the server and the client and each page is its own stateless entity. Technical limitations, especially with Internet bandwidth, limit the possibilities of a pure HTML approach. I worked with the jQuery JavaScript library; a system that facilitates animations, DOM manipulation and XHR (XML HTTP Requests). XHR lets us load data from the server and insert it *within* the current page. jQuery's asynchronous nature was particularly useful because it prevents long-running operations from blocking other functions, so many things can run concurrently without collisions.

The most challenging detail was working out how to display results as the user selects filters. Results are loaded and shown in-line, but there is a possibility that doing so will alter the flow of the page in a significant way, disorienting the user. That can also be a problem if a user unkowingly makes many changes in a short time, for example typing into the package name filter.

I solved this by creating a separate object that actually queries the server for results with the selected filters. Each time a filter's value is modified, that change is pushed to a queue of queries and a timer is set. When that timer finishes, the final query string is sent to the server and the new results are loaded. A two-stage loading indicator shows first that Harvest has seen the user's new selection, then that results are loading. It provides immediate feedback when the user makes a selection and minimises disruption when results begin loading. At any point — even as new results are loaded — packages can be expanded or collapsed to show more details and those changes persist. Harvest intelligently stops, starts and extends queries as the user makes new selections instead of queuing up multiple operations and forcing the user to wait for each one.

Conclusion

My work with Harvest covered a broad spectrum, but each component related in an interesting way. The project furthered my understanding of the delicate interplay between user interface and performance — both real and perceived. The hands-on experience, as well as the Ubuntu community's help and feedback, was indespensible for developing my understanding.

The project took about 180 hours from late May to mid August, 2010. All of my finished work is available at launchpad.net/harvest and I kept a regular journal at dylanmccall.blogspot.com/search/label/GSoC. The current version of Harvest is running at harvest.ubuntu.com. Feel free to contact me at dylanmc-call@ubuntu.com.