GumpPython

The Idea

Current Gump is a bit difficult for most people to develop: one needs to know java, XSLT and shell scripting. So Sam proposed to write a new implementation of the same basic idea, this time using Python. Sam's original random thoughts on this are at

http://marc.theaimsgroup.com/?l=alexandria-dev&m=105024330711316&w=2

ensuing discussion is interesting 😃

The Code

in development inside the gump CVS:

http://cvs.apache.org/viewcvs.cgi/gump/python/

current output:

http://lsd.student.utwente.nl/gump/

Getting started

Gump has two primary usage interfaces:

- Graphic User Interface (human users)
    - This leverages wxPython's wxWindows
- Command Line (human and automatic users, e.g. cron)
    - This provides the basics functions of existing gump (gen/update/build)

Graphical User Interface: Get started with view.py

Key point is that where most of the action is at the moment is in gumpview.

- install 'regular' gump as per http://gump.apache.org/traditional/usage.html
- install Python 2.2
- install wxwindows
- open a shell and do or something equivalent:

```
SET PYTHONPATH=C:\gump\python
CD /d C:\gump\python
python gump\gui\view.py -w ..\myworkspace.xml


cd /gump/python
export PYTHONPATH=`pwd`
python gump/gui/view.py -w ../myworkspace.xml
```

Unless optimized (a switch not currently exposed) each time you run it, it takes time to download all the external project definitions. Subsequent runs load in seconds.

If you select a project, you can walk the project dependencies in both directions, see what prereqs aren't installed, see the actual classpath and properties used in the ant builds, and see what jars a project produces. One pane initially shows the fully expanded XML project definition (implicit dependencies added, properties resolved, etc).

You actually can run a build from this tool. A typical scenario is to select the project that you are interested in, check the prereqs and dependencies tabs. If a dependency failed, you can click on it to go to that project. Press the 'run' icon to build it. If it succeeds, press the 'back' button to return to the project you started with and repeat.

More detailed information on what is being done can be found in the console window which you can access by pressing the console icon.

Command Line Tools: Get Started with update.py and build.py

- install 'regular' gump as per http://gump.apache.org/traditional/usage.html
- install Python 2.2
- open a shell and do something similar to:

```
SET PYTHONPATH=C:\gump\python
CD /d C:\gump\python
python gump\build.py -w ..\myworkspace.xml all


cd /gump/python
export PYTHONPATH=`pwd`
python gump/build.py -w ../myworkspace.xml all
```

Note: There are similar tools w/ similar command line arguments:

- gump/check.py – check a workspace
- gump/update.py – update project(s) from source control
- gump/integrate.py – does "the works" check/update/build/statistics/document/nag

Note: The command line arguments accept a regular expression that resolves over project names **NOT** module names. "all" is converted to the regular expression "*". This is a new feature in Python Gump, and a mild divergence from traditional gump usage. When (say) updating a "list of modules" the utilities calculate the modules containing the requests projects, and hence translates for the user.

Automating the "integration process" using integrate.py

- install 'regular' gump as per http://gump.apache.org/python/usage.html
- install Python 2.2
- open a shell and do something similar to:

```
SET PYTHONPATH=C:\gump\python
CD /d C:\gump\python
python gump\integrate.py -w ..\myworkspace.xml all


cd /gump/python
export PYTHONPATH=`pwd`
python gump/integrate.py -w ../myworkspace.xml krysalis-*
```

This tool is intended for use from cron, or similar scheduling software. It does:

- Reads the workspace and project information
- Performs updates (from source control)
- Performs synchronization (into build directory)
- Performs builds
- All results affect the per module/per project "state"
- State and "work" is maintained in memory and files
- Statistics (FOG Factor and others) are calculated from states & updated
- Once complete the run is "documented" by creating xdocs (for Forrest)
- Forrest is launched to convert this documentation to a site
- Nag e-mails are sent as appropriate.
- Atom and RSS feeds are generated.

See: http://lsd.student.utwente.nl/gump/

Developer Internals

1) XML Model

One of the more elegant aspects of this development is the approach of turning SAX events into an attributed object model, and vice verse. The 'target' classes are self-describing (containing the schema information for things like "repeating sub-elements ought be created using a list of class X", and so on.

For more details see gump/model/*.py and the "engine code" in gump/util/xmlutils.py

2) Context Tree

A "parallel" tree of "associated context information" such as:

```
GumpContext:   ----&gt; Workspace
Contains N of "WorkItem" (e.g. launched Forrest)
Contains N of:
    ModuleContext: -----&gt; Module
    Contains N of "WorkItem" (e.g. launched cvs to update...)
    Contains N of :
           ProjectContext -----&gt; Project
           Contains N of  "WorkItem" (e.g. did a sync, did a "build")
```

For more details see gump/context.py

NOTE: The attempt here is to **NOT polute the object model** with runtime information.

3) Statistics

A simple DBM database of statistics (per project name) is kept containing:

- Number of **Successful** builds
- Number of **Failed** builds
- Number of builds not attempted due to **Prerequisite** failed builds.
- **First date** of first successful build.
- **Last date** of last (latest) successful build.

For more details see gump/statistics.py

4) Documentation

The context tree above, plus statistics, is serialized to a set of xdocs

Tabs: "Main" – the build output grouped by modules, "Stats" – generated statistics, "XRef" – cross reference information. This is a work in progress, and only the first two are attempted.

NOTE: The site.xml/book.xml and skin information are not yet provided.

For more details see gump/document.py

5) RSS

The context tree above, plus statistics, is serialized to an RSS|Atom feeds.

For more details see gump/syndicate/rss|atom.py

Futures and Plans

Here's a list of cool things-to-do:

- put this stuff in some kind of tracker
- document the dynamism in the GOM completely (ugh. Prototype OOP always makes my head hurt)
- look at unittest.py and write tests
- incorporate a callback/interceptor style setup which sends output and info to cheetah or something else (ie you plug in gen.py or cheetah.py into build.py)
- install stuff on lsd.student.utwente.nl
- do some rough speed comparisons
- support <maven/>
- investigate running gumpy with jython, perhaps calling ant in-process (might be a lot quicker; no need to start up JVMs all the time)
- refactor to be more OO and less function-oriented, figure out an extension/plugin setup
- integrate functionality of nag.pl, gendoc.pl, etc.
- more refactoring (no doubt)
- create a gumpy release which works like:
    - unpack (with GUI)
    - ./first-time-wizard.py (with GUI)
    - ./run.py (with GUI)
- create a gumpy rpm which enables

```
rpm -i gumpy-0.3.5.rpm
gumpy install-service
gumpy build-all --result-email-to=gump@jakarta.apache.org
gumpy clean
```

- figure out how to make installing binary packages a lot less difficult
- take a look at POM <-> GOM conversion
- some basic docs (mainly usage and differences from the current gump)
- improve usability of GUI and document it
- integrate/support make and/or automake and/or non-java projects

---

More on Python

None of the existing gump developers are python experts; but it is turning out to be a neat tool. Interesting links:

- http://www.python.org/
- http://www.jython.org/
- http://docutils.sourceforge.net/
- http://www.emacs.org/ or http://www.vim.org/ (no Free python IDE currently available, so you want to use a real editor 😃)
- http://sourceforge.net/projects/pyeclipse/ works fine as a simple *.py editor for Eclipse users
- http://www.mindview.net/Books/TIPython (I'm a Bruce Eckel fan)