Vazels GUI Project

Vazels was a Masters Project undertaken by Angel Dzhigarov last year. He developed a solution for managing comprehensive test suites over a distributed network. The running system has at its head a Control Centre, which administers several groups of remote hosts. From the Control Centre, a user can start or stop the experiment, configure a set of tests for remote hosts to carry out, and collect data from the running hosts in the network.

The user can also perform certain administrative tasks, such as configuring which tests should run on which hosts and how many of each type of host there should be on the network. We, intuitively, call a group of hosts which are configured identically a Group. A Group consists of a number of remote hosts, and specifies what the hosts it contains can do. It does so by means of a Workload, which tells the system about snippets of code which will be stored on the hosts of the group and named Actors.

Broadly speaking, the hosts fall into two categories: Vazel machines, and SUE machines. Vazel machines hold user-created tests, which are invoked at runtime and will report back information. SUE stands for "System Under Experiment" – it is the system that the user is trying to test out using the distributed network. It is worth noting that a host can be both a SUE machine and a Vazel machine – that is, it is both running the System Under Experiment, and running a set of user-created automated tests.

At the moment, a user must utilise the command line to start the Control Centre, specifying the names and sizes of all the groups, as well as other configuration options (for security, and regarding the mechanisms that Vazels uses for internal communication). They then have to use a separate command line client to specify the workload for the groups, and they must then copy files into a newly created directory structure, matching up their chosen group names to the right folders. The conclusion is that the user is forced to spend time engaged in setup tasks, which can scare users away from trying out a very capable and helpful system.

The difficulty in setting up projects and visualising the concepts involved provides a large part of the motivation for this GUI project. We aim to provide the user with an intuitive interface to set up an experiment, without having to type on the command line, or deal with setting up tools such as Siena or an RMI Registry on remote machines (both of which are currently involved in the setup process). We also aim to provide a useful way for the user to gather results about their running experiment, and monitor its status in real time.

Key Requirements

We have chosen to break the goal of providing a GUI to Vazels into two Phases:

Setup Phase

In this stage the user will configure settings for Groups, the number of hosts in them, and which actors and workloads will be loaded onto the hosts in a given group, as well as where the SUE will be situated. The user must be able to

- Set up Security – the user is expected to provide private keys for the system to enforce a Private Key security model (and to enable zero-config SSH at runtime)

- Specify crucial constructs of the test, which together are:
  - Workloads – these define the tests which will be performed by remote Vazels
  - Actors – the snippets of code that are invoked by workloads
  - Groups – the configuration of which hosts will be given which workloads. The user will specify the number of hosts in a Group, the workloads performed by hosts in that group, and any additional code to be run on those hosts (e.g. computers in a Group may also run the SUE).
- Upload the files required by the test system (the SUE, any code snippets required as Actors, workload specifications, and any other files required) without having to consider the directories the files will eventually end up in – the system's back-end will deal with administering these details.

Running Phase

In this stage the user will be able to monitor their running experiment. They must be able to:

- Display which hosts are online or offline at any given time, categorised by Group, with the ability to have a Group of Vazels run a workload that has been assigned to it.
- Have the Control Centre collect data from attached hosts, then display the output of their tests. Since tests are defined by the user, it is difficult for us to know ahead of time what data might be saved, so at its most basic level this might just consist of a raw dump of the data.
- Shut down the server.

Other Key Requirements

- The user must be able to administer a Vazels Control Centre that is not running on their local machine – one of the target platforms for the Vazels project is PlanetLab. The GUI must provide a useful interface to a remote user.
- Because of the previous requirement, the GUI will be provided as a Web interface. The interface must run on at least Firefox and Chromium.
- The back-end for the Web interface must be easily configurable by an inexperienced user. It is required only to run on a *nix platform, since that is the only platform Vazels can deploy on.
- Since the Web interface is providing remote users with potentially powerful access to settings on a remote device, there must be reasonable security restrictions.

Extensions

- First and foremost, we would like to provide the user with a more interesting display of the data that is collected by the system in running tests. If possible, the user will be presented with a more useful representation of the data, e.g. a graph of how a given variable changes over time (in the case of a number being recorded), or a timeline of the different values a string took, at which times.
- Prefabricated tests – currently all tests are user-written. We think it would be useful to have some pre-made tests that a user can “mix in” with their existing tests and workloads. These tests might provide data such as open network connections, or current CPU usage. Since we already know the type data returned by these tests, we should be able to provide the results in a compelling user interface.
- Visual “point and click” workload building. We would like to use code-browsing techniques on the actor code to allow the users to define workloads by “selecting” actors in the GUI, rather than by hand-coding a workload specification.
- Saving. Most users will want to run the same suite of tests more than once, and we’d like the user to be able to save their setup and come back to it later.
- Visual “point and click” group configuration. We would like the user to be able to have a visual display how their groups are put together.

Choice of Development Method

Code testing and version control

We have decided to use a distributed version control system: Git. We are using a peer-review model for committing changes, with “topic branches” for all new features. Each member of the group is assigned two “code reviewers,” whose job it is to check through their code before it is approved for inclusion in the Master branch. We hope that this will first help us to catch errors, and second encourage a shared sense of responsibility (and ownership) for the correctness of code.

We are using Trac for issue tracking and other project management tasks. It also has a built-in wiki which we are using as a team knowledge hub. We preferred this solution over others as it gave us a good mix of flexibility, ease of use, and ease of configuration.

Code testing is an integral part of any development process, and part of the review process is to ensure good test coverage. Test-driven development is being employed where possible, with the caveat that unit tests are often impractical when producing a user interface – usability is a subjective issue. For this reason, we are using a system of “user stories” and feedback from our “client” (Professor. Wolf).

Software development model

We are combining agile practices (such as regular meetings and efficient knowledge sharing) with a strict organisational program necessitated by our busy schedules around the rest of our studies. We aren’t having the daily meetings recommended by many Scrum manuals, instead preferring to have slightly larger work packages to allow individual flexibility and personal responsibility for time-management. This will be subject to change as the project progresses and our other commitments vary throughout the term.

Because of the modular nature of parts of our project, methods such as Extreme Programming should be easy for us to employ where appropriate – as is already the case in the development of our REST server, where a programming pair is currently writing the code for our Settings Management module.

Required IT infrastructure

Because we are building a GUI for a distributed test management system, we require a test-bed of host machines to run our tests on. Fortuitously for the project, the computers in the Department of Computing labs are an ideal candidate for such a test-bed, with the required SSH setup already in place. We are also using a remotely hosted Git repository and Trac system (as mentioned above). We may use the PlanetLab test-bed to test our system later in the development cycle, but have no specific requirements from the Department of Computing at this stage.

Project Management

As a group of five individuals, we have a simple management structure, with James assigned to the role of Team Leader to keep administrative focus consistent, and ensure the project maintains a unified perception of goals.

As has already been mentioned, we are employing an issue tracker to help with time management and accountability. Individual team members are already beginning to specialise their knowledge of the extensive project we are building upon - we are actually taking measures to reduce the risk of over-specialisation (e.g. with the wiki and knowledge sharing already mentioned), but it is natural parts of the team take over more responsibility for the areas they work most on. The current breakdown of the main workloads features:

- James managing administrative tasks, and ensuring system coherence,
- Andy managing the RESTful back-end server,
- Andreea managing the front-end Ajax GUI,
- Simon managing information aggregation from tests that have been run,
- Chris managing communication between the RESTful server back-end and the Control Centre.

At this early stage in the development process, this breakdown is sure to change - perhaps drastically - by the end of the project. All team members will have at least some involvement in all aspects of the project, and we feel the code-review process we have put into place will help this.

First Iteration Plan

First iteration requirements

We have basic requirements for a first iteration of the project, with a delivery milestone set for 01/11/10. The User must be able to:

- Configure the topology of the test setup (How many groups? How many hosts in each group?)
- Add workloads to groups (using hand-written .wkld files after the group configuration has been submitted to the server)
- Start and stop the server.
- Run individual tests
- Take test snapshots and record the results on the server end
- These snapshots will be arranged into an XML or JSON tree server-side. This is a pre-requisite to later goals.

The interface will be an HTML front-end, which will send AJAX requests to the RESTful server back-end Groups will be configured using a single form, and sent to the server using a simple "form submit" button. Instructions (start/stop test, take snapshot, etc.) will be sent using anchors within the page which will not cause the browser to refresh.

Breakdown of responsibilities:

- Configuring the topology of the test setup: Andreea will be preparing an HTML page containing the appropriate form controls to assign names and sizes to a user-selected number of groups. Once submitted, the page will make a request to the server, informing it of this change.
- Andy will prepare a REST receive-URI in order for the server to receive this update. He will deliver the capability to communicate to the server by Friday (29/10/10), along with the server's capability for it to save.
- Ability to assign a workload to a group will be implemented by Chris in time for the milestone on Monday (01/11/10).
- Andy and Chris will need to liaise with regards to deciding on a REST receive URI for the submission of the workload file. James will take responsibility for assigning the workload to the correct group server-side once it is submitted
- Andy will provide REST URIs on the server which, when called with POST requests, will start or stop the server. The server-side code for this is trivial but non-blocking so is due on the milestone on Monday
- Simon will be providing the user with the GUI buttons that effect these commands, in the same timeframe.
- Andreea will handle the user-interface for running individual tests (starting workloads). This will involve giving the user a choice between the already submitted workloads to be run.

- Andreea and Andy will have to liaise by Friday (29/10/10) on a REST URI pattern for this submission.
- Snapshot handling is a large task and the back-end will be handled in its entirety by Simon. Code submission for this should be on Sunday evening so that a code review can be done in time for the milestone. James will handle the user interface task for this, which is trivial.

Potential Risks:

- Failure to meet these short time deadlines. Realistically, we will be hard-pushed as a team to meet all these goals by Monday. To mitigate this, group members with smaller workloads (or who finish early) will have to help others to get their work done on time.
- Failure to achieve expected results with the GWT platform which we are using to produce the client-side interface. Using Google's GWT platform, we are hoping to harness the power of statically-typed Java, which in our experience is a much more productive environment. As always though, adopting a new toolkit has a set of risks and unexpected problems.
- At this stage, we have yet to write any tests. We are aware of this as an issue, and the potential for code regressions. Our review process should help with this, and our current code-base is small enough that the risk should not be great, but as we move forward we hope to achieve much better coverage.

Progress Measures:

Given the short time-frame of this iteration, progress measures may reduce to whether or not deadlines are met. Trac tickets will help us with this, and we hope to evaluate the progress that has been made since the beginning of the project at the end of this iteration.

By the end of this iteration, we will have a system that allows the most basic setup of the Command Centre to be carried out through our user interface. From there, the user will still need to finish putting everything in place before their test can run, and it will be up to them to retrieve the collected results from the server.

Draft Schedule

The time constraints on this project make it worth us setting out weekly goals at this early stage. We will break down the workload into weekly milestones, hoping to deliver a final iteration on 18/12/10. The description with each milestone details what work will be undertaken in the week leading up to it. To summarise:

- We plan 5 iterations in total over a 7 week period,
- We aim to complete functional requirements by 13/12/10 (and to have realised our key requirements by 6/12/10).

saturday, that's keen

- Major requirements should be finalised by 08/11/10, with major changes permitted after that only with considerable justification.
- No new requirements will be considered within the scope of the project after 13/12/10. By producing a number of iterations before this point, and getting regular feedback from the client, we hope that there should not be any such requirements by this point.

25/10/10 – Research & Concept Exploration

- Preliminary research of the Vazels project, and a good understanding of the concepts involved by all members of the group. A good understanding of the motivation and some of the requirements for a GUI to the system.

01/11/10 – Key Requirements

- Key project requirements finalised with the project supervisor.
- **1st iteration complete.** A basic "setup" solution, to help the user administer groups and workloads. The user will still have to manually provide the system with required files (such as Actors, and the SUE).
- A pause for revision and reflection on issues so far.

08/11/10 – User stories, and server-side management of [illegible]s

- Key user stories completed, and discussed with client in some detail.
- Major changes to requirements will be considered only with adequate justification.
- Implement a system to allow the user to fulfil the security requirements of the Vazels system easily (or removing the need from the user, and automatically generating a new SSH key on a per-project basis).
- Server can now build an abstract model of Groups, Actors, and Hosts. The user can provide the necessary files via the GUI, and no longer has to manually configure server-side file locations, etc. When the user wishes to start an experiment, the Web interface will manage the back-end tasks for them.

15/11/10 – Setup complete: leaving the command line behind

- **2nd iteration complete.** Functional requirements of the setup phase are now met (changes to requirements notwithstanding).
- Crucially, the user no longer has to touch a command line until they are ready to deploy their test onto the machines taking part in the test.

22/11/10 – Feedback and data display

- The user can now retrieve test data from the server (via AJAX request, meaning that they do not have to refresh the web page). A basic display of this data should be implemented. If reasonable within the time-frame, data will be presented on either a static graph, or a time-line. *[Extension: Depending on how difficult this actually proves to be to implement, we will decide on the feasibility of a more rich data display at this point, as well as a time-frame for putting it into practise]*

- Take this opportunity to explore the feasibility of some of our other extension tasks. Specifically, investigate the performance overheads (and implementation details) for the "Prefabricated tests" mentioned previously (e.g. monitoring CPU usage).

29/11/10 – Finish key requirements

- **3rd iteration ready.** Server security model fully implemented. Functional requirements of running phase ready. User can run tests and display data collected from them. All key requirements should now be a part of the project. **Feature Freeze.** No new features will be started after this point until after the next milestone.

6/12/10 – Client feedback and requirement revision

- Finish implementing incomplete tasks from 3rd iteration. Revise project requirements, and decide on feasibility of extensions.
- If finishing features from 3rd iteration and bug testing goes better than expected, begin to investigate possibilities for implementing extensions.
- Client-centred testing. Ensure the client is satisfied with work so far, and address any concerns.

13/12/10 – Document, test, and finalize

- **4th iteration complete.** All functional requirements now realised. **Re-freeze features.**
- Ensure good test coverage. In cases where it is difficult to write tests (e.g. UI elements), find other ways to validate code.
- Ensure good documentation coverage.
- Ensure the Wiki forms a complete and useful picture of the development process – and of the product as it currently stands. Ensure information is accurate and up-to-date.
- Ideally, a new team of developers should be able to continue our work after we finish – given access to our code repositories, our documentation, and our Wiki.

18/12/10 – Final Iteration

- **5th iteration complete.** The quick turnaround between this iteration and the previous one should be possible by this point, where all code changes are bug-fixes.
- No remaining "Blocking" or "Major" bugs.