Specification for Redmine-based Testbed Portal

Table of Contents

Revision History:

Version	Date	Changes
1.2	15-Sept-2009	RD&R now in scope; AM not running in the server
1.3	16-Sept-2009	SVN->git; add version column in fig 2; rectification 15% not 20%; add test cases for milestone 1; detail the resource description of the experiment (milestone 3)
1.4	23-Sept-2009	Added alternate sprint 2
1.5	29-Sept-2009	Merged milestones 1 & 2 to single sprint; Merged alternate sprint into sprint 1; defined terminology, and the AM-Portal interface.
1.6	30-Sept-2009	Clarifications from document review (Jolyon)
2.1	02/11/09	Add 'type' attribute to the XML upload; change layout of properties and show only link to logs; use GIT per project; follow reference URIs in script display; basic script editor (not ruby aware); tagging scheme for scripts; user creates name for scripts; remove sprint 2 filtered list as it was done in sprint 1; git revisions will be tagged by a convention; user friendly revision will be the tag name.

PART 1 - SUMMARY & BACKGROUND:

NICTA has an existing network testbed infrastructure, at ATP Sydney.

The software for managing network experiments ("OMF") is provided by NICTA and others, and documented at `http://omf.mytestbed.net/` (a Redmine site)

An explanation is found at
`http://omf.mytestbed.net/wiki/omf/An_Introduction_to_OMF`

In summary:

- experiments are scripted in ruby language (with extensions)
- 'Google Calendar' is currently used to reserve a testbed (a set of network nodes)
- at the reserved time, the user logs in and manually starts the experiment run
- the AM (aggregate manager) runs the EC (experiment controller), which orchestrates the nodes according to the experiment script
- results are found in a SQLite database on the AM server
- one experiment may be run many times, optionally with modification of the script and optionally with variant parameters ("properties") passed to the script

The current system is missing some higher level management functionality:

- a reservation system that does not rely on the user picking a timeslot in the calendar
- automatic running of the script at the reserved time (which can be out of office hours)
- a means to track the scripts and the result of the runs.

A new Portal web service is required to create & edit scripts, manage schedules of runs, and the results of those runs. The Portal is primarily a data repository. It does not directly run the scripts. They are run indirectly by the AM which queries the Portal for the upcoming schedules.

Refering to figure 1 – Block Diagram of Proposed System:
It is proposed to use another instance of Redmine for this portal, with a plugin, and a new process RD&R to manage the scheduling and calendar. These all run on the same server.

For consistency of terminology, the following terms are used:

Script	The experiment script, written in ruby
Version	The version of the script e.g. r73 if using SVN, or other means to identify a git version
Properties	A set of initial properties (name:value pairs) which are passed in to the script for a given experiment run
Run	One instance of running a given Script/Version/Properties
Session	A session is a timeslot reserved by a user, during which they may run experiments manually, or schedule them for automatic running. A session has a start time/date, and a duration

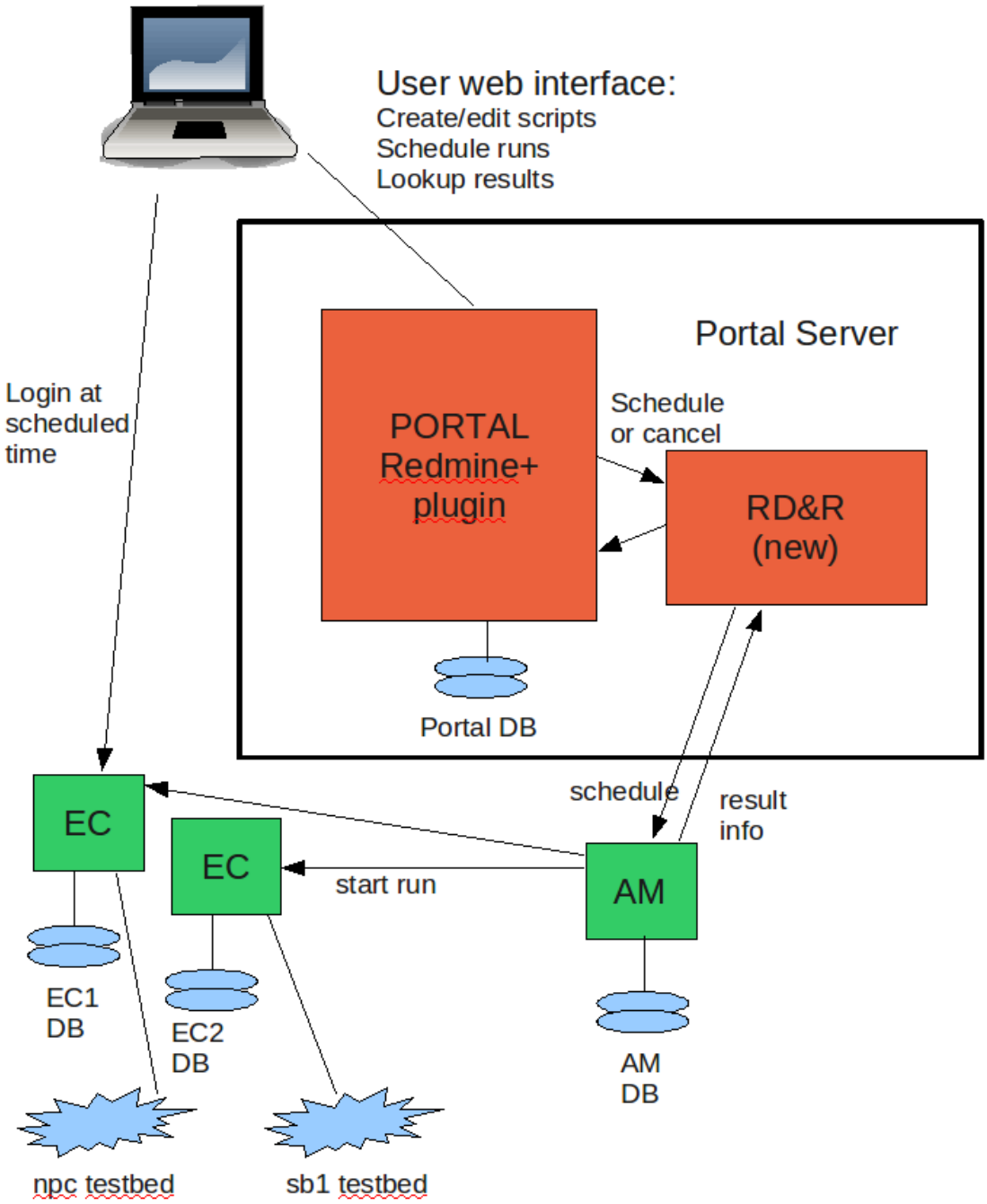

Figure 1: Proposed System

PART 2 - REQUIREMENTS:

GENERAL:

- The portal will run on a public server. The subdomain omf.mytestbed.net is currently pointing to an instance of Redmine at a hosting company, and svn.mytestbed.net for its code repository. The new instance will need to co-exist on the same domain name, using subdomains. The suggested subdomains are:
  ```
  www.mytestbed.net for the redmine portal
  git.mytestbed.net for the repository
  ```
- It may be on a different server or the same server as the omf.mytestbed.net website: NICTA will set the DNS pointers accordingly
- The server OS is Ubuntu 9.04.
- The preferred web client is firefox, on Mac OS, Linux and Windows. Internet Explorer does not need to be tested, but the IE user should get a message that 'Firefox is preferred' **if** it is really broken on IE.
- The additions to Redmine need to be modular &/or packaged to permit upgrading Redmine, ruby & rails. InContext to advise, and write this up in the documentation.

DELIVERABLES:

Incremental development and delivery is used, with 3 delivery “sprints” as described below.

The script page (figure 4) and result page (figure 2) capture the key elements of the Portal user interface, although they will not be fully populated initially. These are not the only screens of the portal, but the important ones.

Sprint 1: Basic Redmine Setup with code repository.

- InContext to install & setup Redmine on the server provided by NICTA, including the expected 3 database configuration (development, test & production), deployment tools etc
- GIT is used for the script source repository. The database (PostgreSQL) is used for other attributes, and the user database
- All new users need to be approved by an administrator.
- Users require passwords to login.
- User creation needs to be logged, as well as subsequent login and logout actions.
- Incorporate a captcha in the new user request to minimise robot requests (or suggest an alternative). If a captcha is provided, it could be donated to the redmine project, rather than be a project special for NICTA.
- An administrative page shows all user logs.
- A preliminary database is to be setup. It should be viewable and editable using default RAILS views. It will **not** appear like figure 2 yet, but figure 2 is a reference to show what information is required in the database.
- The 'Types' attribute should also be set up, with administrative access to add new Types. A

link to Type data view/edit appears on any administrator's home page. This is a permanent feature.
- The Project attribute is to be the project as defined in Redmine, administered via the usual Redmine tools.
- Database migration is to be set up & used from the outset
- A design document is to be started in this sprint, in the style to be continued for subsequent sprints. It should include a section on administration, describing how to start/stop the application, how to migrate database changes, how to deploy and roll-back development through to production.

Test Cases for Redmine Setup:

1. Unstructured test of the redmine installation: check that the basic administration functions operate e.g. add a project; wiki and other usual tabs are operative
2. Check that git is linkable to a project and browsable from redmine
3. A user attempts self registration, but mistypes the challenge/captcha: fails & log is created
4. A user attempts self registration successfully. Administrator approval is required. Log is created.
5. New user logs in: log is created.
6. Script data and Types data can be viewed and edited by an administrator. Links to these should appear on the home page of administrators.
7. Documentation explains start/stop/migration/updating/deployment. Documentation describes the plug-in addition, and where to find the code.

Results Page

The work items have been re-ordered now. The idea is to begin with results capture. Researchers will operate the experiment system manually, as it is done today, but the Portal will capture the results and log file information, and display it on a page.

This is a much needed feature, because currently the researchers have to write down the results identification and location information manually, and risk losing track of the results, script source etc.

With the new focus on the results collection in sprint 1, the viewing and editing of scripts is moved to a later sprint. However, it will still be necessary to design the database (or redmine equivalent features) to accommodate the eventual Scripts and Runs data.

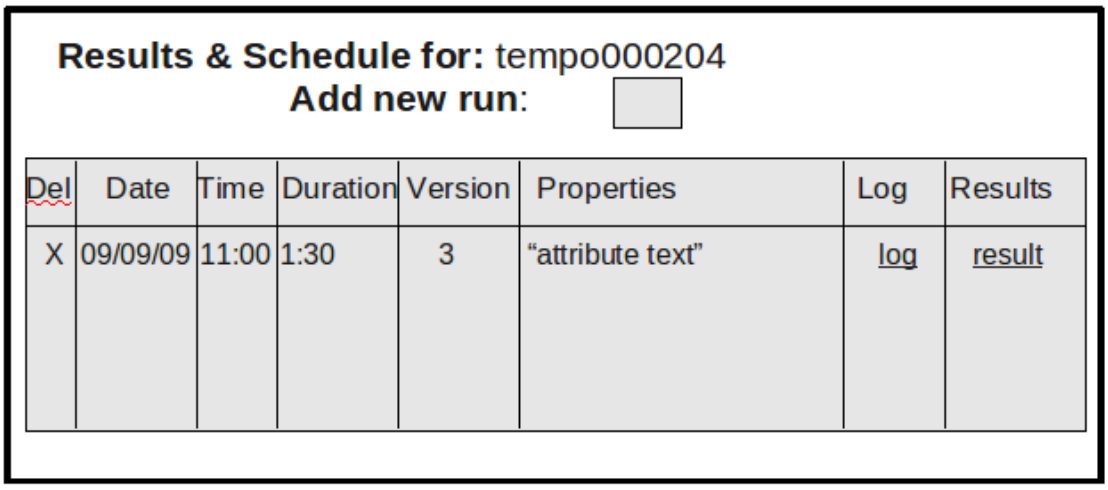

Figure 2: Results Page

- A script has a unique identifier, a location in git, the script itself (ruby code), and some other attributes:
  - user name (Redmine ID)
  - type (wifi, 3G, ethernet, DSRC ..) from drop down
  - free text description
  - creation date
  - 'resume' flag (resume the same database for subsequent runs, or not)
  - 'batch' flag (run automatically if set, or manually if false)
  - project name : drop down of the Redmine projects for which the user is a 'developer'
- A research can run a script manually, using a session reservation in Google Calendar, exactly as it is done today.
- At the end of the run, however, the AM uses a defined API to advise the Portal of the result data.
- The locations of the results, the log file, and the graphs, are all found in the AM database, which is external to the Portal server. The AM will push the information to the Portal via an http POST of an XML structure. The XML format is shown in figure 3.
- For security, the AM-Portal communication uses SSL (https) and a certificate for client authentication.
- Note that the AM is providing some of the data content (script code & log content), not just URIs, because we cannot be sure that NICTA or any of the other parties will allow the Portal to access the data at the URIs
- RD&R puts the information into its database, indexed by script name, using the information provided from the AM over the API.
- If the script is not already in the script database, an entry is created.
- The Portal will populate the database as necessary when advised of a run result. For example, the script may not exist yet in the local database. It's URI may be to an external system. The Portal is not able to query the external system. It only knows what it receives

via the XML message. Initially, this may include the actual text of the script, as a temporary measure for sprint 1.

- The user can now find the results page (figure 2) by filtered search. Filter (at least) by project, userid, script name.
- When a run is complete, a link to the **log** file appears on the results page. This should use 'web2' methods for dynamic update, and a hover-popup showing the log contents.
- The **results** link simply shows the URI to the results. They are not accessible from the Portal, but the web user may have access to them.
- The results page should be available on a URI which a user can put into a Redmine wiki page, which they will use as their 'lab notebook'.

```
<experiment-outcome>
  <project>my-project</project>
  <id>unique-experiment-id-string</id>
  <userid>userid</userid>
  <userauth>authentication-token</userauth>
  <start>2009-09-28T15:28:49Z</start>
  <duration>3.141592653589793</duration> <!-- About 3h09m; could also use xmlns
duration spec -->
  <tags>
        <tag>bittorrent</tag>
        <tag>p2p</tag>
        <tag>algorithm-1</tag>
  </tags>
  <script>
        <uri>git://script-uri.com/repos/project.git/path/to/script</uri>
        <revision>6994be1c55bee59f50dd01141f0801d045576fdd</revision>
        <type>wifi</type>
        <source>
# A) Define the 'source' group, which has the unique node [11]
#   Nodes in this group will execute the application "test:proto:udp_sender'
#
defGroup('source', [1,1]) {|node|

    node.prototype("test:proto:udp_sender", {
       'destinationHost' => '192.168.0.2',
           'localHost' => '192.168.0.1',
           'packetSize' => 256, # in Bytes
           'rate' => 8192 # in bits/sec
           ...
        </source>
  </script>
  <script-src>
  </script-src>
  <properties>
        <property name="max-rate" type="xs:decimal">10.0</property>
        <property name="max-connections" type="xs:integer">50</property>
  </properties>
  <resources>
        <resource>...</resource>
        <resource>...</resource>
  </resources>
  <measurements>
        <!-- The URI's don't necessarily point to the filesystem; could be a
web-service instead -->
```

```
        <!-- The contents of the URI's could be sent in subsequent messages -->
        <!--
          The <metadata/> element refers to the contents of the
          _experiment_metadata table in the database.  Also possibly the
          _senders table should also be sent, because the experiment
          measurement tables can't be completely understood without it.
        -->
        <database>http://measurements-
uri.com/path/to/project/database.sq3</database>
        <log>http://measurements-uri.com/path/to/project/log.txt</log>
        <logdata>The actual log file contents appear here because the portal
cannot necessarily access the URI</logdata>
        <metadata>http://measurements-uri.com/path/to/project/database.sq3/sql-
query</metadata>
  </measurements>
</experiment-outcome>
```

Figure 3: XML Result Structure from AM to Portal

- Testing of this sprint can be done initially with some sample XML files provided by NICTA, using the 'curl' command line utility to POST them to the Portal.

Sprint 2: Add the Script view/edit page

- A user can jump to a list of all their own scripts sorted by Script name.
- The user has the ability to add a new script, or edit an existing script.
- If adding a new script, there is an intermediate screen where a drop down list of projects is shown, to select the project (cannot be changed later), and a selector to choose the initial script copied from an existing script which may be from another project, such as the 'tutorial' project (navigation required).
- Then jump to the script page (see figure 4) where details can be entered or edited.
- There is no 'delete script' function.
  In practice, scripts are run in a project, which has a certain lifetime. Eventually the project is only of historical interest.
- The unique script name is entered by the user. Redmine will create a new branch in the project GIT for the script, if it is new. The script name is (silently) enforced by the Portal to have only alphanumerics: no spaces or punctuation.
- A new script could arrive via the XML interface, or the user interface.
- The initial script can optionally be copied from an existing script (browse button, limited to a particular directory on the server, and sub-directories).
- A basic web editor is embedded into the page, for editing the code. The script is saved in git. The window shows the most recent version by default, but a selector allows the viewing of another version (or maybe this is just the redmine-repository viewing feature)
- At any time, a script can be edited. Code changes are committed to git. It is assumed that only the project members will edit the script, along a single 'main-line' (no fork). If another person edits and commits, the last commit wins.
- GIT does not have the concept of a consecutive revision number (like SVN “r2” etc). Revisions will be tagged using a convention. This tag will be used to give the researcher a simple view of the version.
- InContext to suggest a scheme, e.g.:
  - the number of commits on a path [git rev-list br1..doc | wc -l]
  - the timetag of the commit
  - some other naming scheme
- This tag will be used to give the researcher a simple view of the version.
- A script may refer to other scripts. It should be possible to follow the reference by clicking on it. The references may be to other projects, if the user has permission for them.
- The Design Document should be updated for the sprint changes.

Some modifications to Sprint 1:

- In sprint 1, the properties were shown as a long text string. In sprint 2, they will be formatted as a table with 2 columns: Name and Value.
- In sprint 1, the actual log text was shown. In sprint 2, only a link to the locally stored log text will be displayed. The user can click the link to see the log text.
- In sprint 1, there was one GIT repository for all project scripts. In sprint 2, there will be one per project.

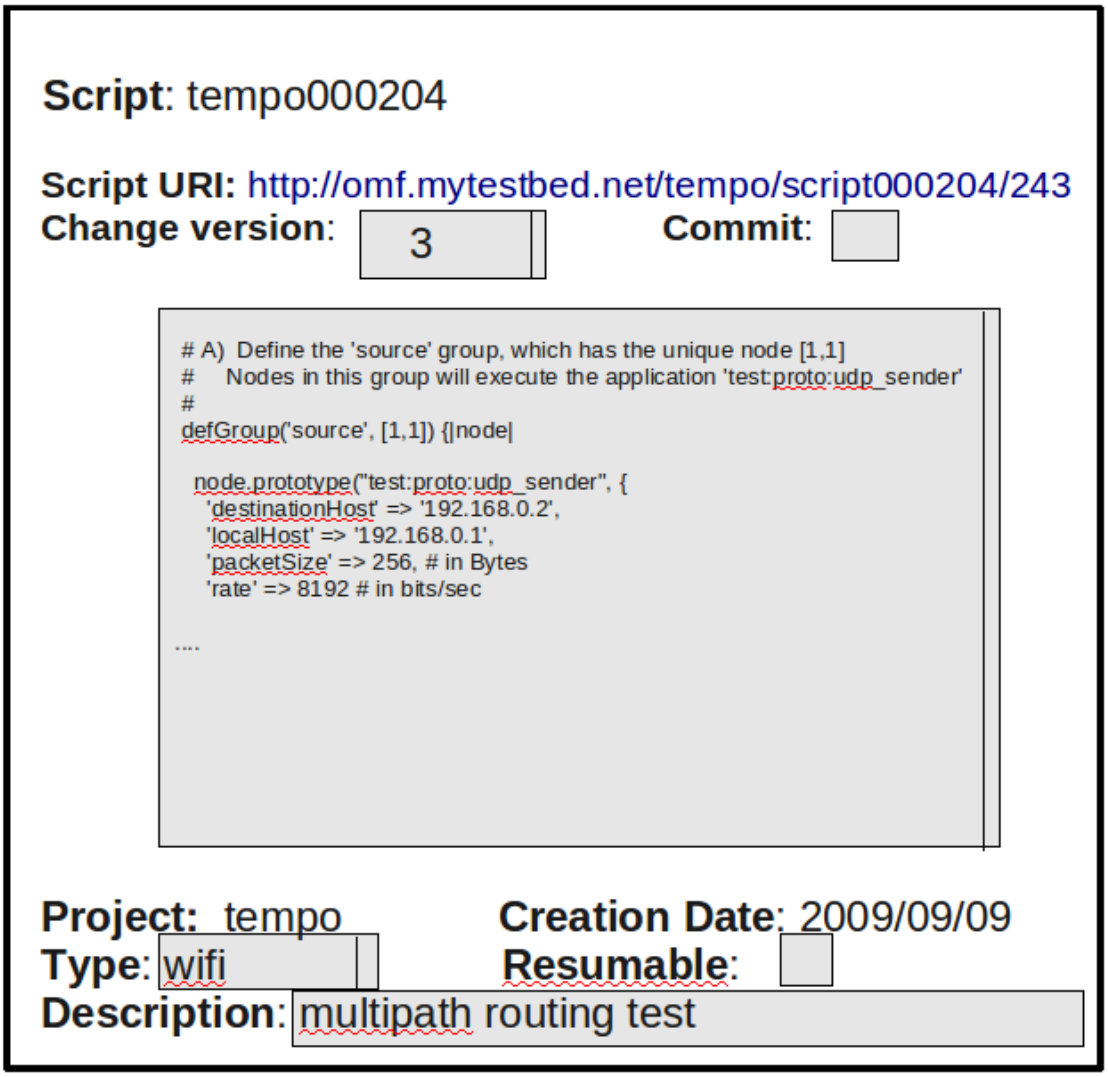

Figure 4: Script Page

Sprint 3: Scheduler additions:

- The user can create the session entries, using a half-hour minimum granularity, or “ASAP”, but still ending on a half-hour boundary. There is no restriction on this: anyone can create session objects at any time, even overlapping ones. The restrictions come into play when a run is scheduled.
- *However, in future (not necessarily in this project timeframe), it is required that an administrator can 'bump' a user off their reserved timeslot and all scheduled runs in that session.*
- A user can schedule runs of a given experiment/version to run at some time in the future (during the session). They can be set for manual or automatic running. The AM will request the schedule, and take care of automatic running.
- Multiple runs can be scheduled for one script.
- Multiple runs can be scheduled within one session.
- User can list, edit, add, and delete entries, but
- PREVENT deletion of a schedule entry if it has already run to completion.
- The scheduler must avoid clash of resource use (nodes, spectrum etc).
  The portal plugin is not involved in the scheduling calculation; it asks the Resource Reservation & Discovery service (RD&R) for the earliest possible date/time which meets the user request.
- There may be **no** solution, in which case the portal must display the Reason text returned from RD&R.
- The interface to RD&R involves:
  1: parameters sent to RD&R:
  - URI to script/version (which contains resource information)
  - The earliest date/time (or 'ASAP')
  - Duration of run (integer, in minutes).
  - Run initial properties

  Note that a user may reserve some nodes without a working script. This is typically done for debugging a new script. However, there will still be a minimal script, which has at least the resource information.
  2: information returned from RD&R:
  - The start date/time
  - A booking sequence number
  - The duration (a copy of the duration from request)
  - An expanded resource description: this a text blob to be saved in the reservation record. It elaborates the resource definition, especially where the user was not specific.
  - Reason code: "OK" or a text message explaining why the booking failed.
  - A public key signature over all the above information.

  Also there is a cancel command, which requires only the booking sequence number and returns “OK” or a reason text if the cancel failed.
- If the reason code is "OK", the booking is made.
  This is an atomic operation to avoid the need to hold a reservation open for a period of time.
  If the user is not happy with the selection, they can delete the booking. The scheduler ensures at least 1 minute delay to the start time, to allow the user time to cancel.

The portal stores the expanded resource description in the booking record. It is a readable text format, which can be read by the user by hovering the mouse over the resource field in the booking list.

- The portal also stores the date/time, duration & run properties string.
- The resources can change as the runs evolve e.g. run it on a sandbox initially, then a larger testbed later. To keep it simple, the user edits the script, and schedules the new script/version.
- The AM needs to be able to find out the schedule of upcoming runs, using an http GET request, and receiving an XML reply. (Needs to be clarified before this sprint.)
  This interface should also have an HTML rendering, so the users can view future reservations.
  The proposed interface is a web interface, providing:
  1. An html table of the list of upcoming reservations
  2. A REST style XML rendering of the list of upcoming reservations
  3. An HTML detail report of a single reservation, by ID as given in the list, or the 'next reservation', if ID is "0".
  4. A REST style XML report of a single reservation, by ID as given in the list, or the 'next reservation', if ID is "0".

  The reservation detail needs to include:
  - the script/version URI
  – the run properties text (which may vary between runs)

Resource Discovery and Reservation System (*INCOMPLETE – needs clarification*):

The resource reservation system presents some unsolved challenges at this time. Therefore this section is expected to change before the commencement of sprint 2.

- a simple resource booking system will be used initially, which will rely on the user to select from a calendar view (which might somehow incorporate a floorplan view)
- The resources requested in the experiment description are concatenated with the username, the reservation date/time and duration, and this is all signed with a public key signature. The result, called a “ticket”, is sent to the user in an email as the proof of their reservation. It is also stored in the portal database, in a reservation record.
- The resource description is extracted from the text of the script/version using text pattern matching. Nodes are identified on an X-Y grid e.g. “[3,2]”. The keywords are highlighted in the following text:

  Here are some examples of requesting nodes explicitly (by X,Y):

```
defGroup('source', [1,7]) {|node|     # node [1,7] explicitly
defGroup('sink', [1,7]) {|node|       # node [1,7] explicitly
defTopology( name , arrayOfNode
  where arrayOfNode can be:
     [2,6]              # one node explicitly
     [2,6],[4,8],[5,5]     # a random list
     [2..4, 6]   # range of X, here [2,6] [3,6] and [4,6]
     [2,6..8]     # range of Y, here [2,6] [2,7] and [2,8]
     [1..2, 1..5]    # a rectangle of 10 nodes
```

Here is an example not specifying nodes explicitly. This case is problematic, because it may appear inside a loop of ruby code! *This is out of scope for the current development.*

```
t.addNode(aNode)    # one more node required, may occur many times
```

- If & when a user tries to access the AM to start a script manually, they must provide the ticket. The AM can then check the resource list, and the date/time to allow or disallow the script to run (NICTA responsibility).
- The Design Document should be updated for the sprint changes.

PART 3 - NICTA RESPONSIBILITIES:

INTEGRATION & NICTA PROVIDED COMPONENTS:

- NICTA to provide the production server, and adjust DNS pointers for the domain, relocating 'mytestbed.net' site if necessary. Provide accounts for InContext.
- NICTA to write the acceptance test cases before each sprint begins, and to run the acceptance test, and send a written list of defects to InContext.
- The Inventory Database needs extensions to track runs by project, script name and version, date, results location, results graph location, & result log location.
- AM and EC modifications are also required, in order to:
    - Request the schedule from the Portal
    - Activate EC instances at the required time
    - Notify the Portal of the results (via the Inventory extension)
    - Validate resource tickets & permit or disallow access
- These modifications may not be ready for the test of sprint 1, so NICTA must provide some sample XML files, so that 'curl' can be used as the test stub.
- NICTA to measure the connectivity matrix between nodes, before a more advanced scheduling algorithm can be implemented.

FUTURE WORK:

This work is out of scope for the current development, but consideration of these points may be of interest in the design stage:

- *AM or EC will in future program the VLANs dynamically, according to the resource list. Static VLANs will be used for the current work. This is not expected to impact on the Portal.*
- *RD&R may in future consider spectrum reservation in addition to node reservation. This should be transparent to the Portal.*
- *It would be good if the RD&R function could provide a web accessible view of upcoming reservations, maybe in a grid of nodes vertically, and half-hour slots horizontally.*
- *The results database may be distributed, in future. The 'location of results' in the AM database might then link to a URI which has active code to merge and graph the results.*
- *An administrator can bump a user off their allocated session timeslot.*
- *A 'resumable' experiment is one where the results of several runs can be concatenated. This is not yet supported by the EC. It will impact the Portal significantly.*

SECURITY:

- (by NICTA) Need a risk analysis, and a list of open ports.
  Currently, this looks pretty simple: port 80 for the redmine web user interface, port 443 for the API, and port 22 for ssh access.
- (by NICTA) Need a backup and recovery plan. Allow for multiple backups, and time to discover problems before the oldest backup is lost. InContext to advise which directories and databases need to be backed up.

PART 4 - DELIVERY, TERMS & CONDITIONS:

The Portal is to be delivered in defined increments ("sprints"), as specified.

Quotation price to be provided per delivery sprint. Each sprint will be treated as a separate order. The quotation is to be provided before the commencement of each sprint.

Acceptance testing: NICTA is to develop the acceptance test cases at the commencement of each sprint, and to carry out the acceptance testing within 5 working days of delivery. Faults are to be classified as Critical, Major or Minor.

Payment is to be made by NICTA at the rate of 85% of each quotation price on delivery of the sprint, and the remaining 15% on the rectification of all critical and major defects (if any).

Delivery Dates: The final delivery to be before 30th November 2009, but preferably earlier.
No exception to the end date is possible, due to expiring source of funds.
Sprints not achieved by this date will not be paid for.

Rectification work must be completed soon after this date. The absolute final date for the rectification work payment is Friday 18th December 2009.

Documentation: Class diagrams, and other code design information (preferably in UML style) are to be provided.
The extent & type of documentation is to be advised with the first sprint quotation.

Source code: Source code to be provided (of course!) and ownership to be assigned to NICTA, free of patent encumbrance, copyright claims etc, such that NICTA can carry out ongoing modification, and potentially to open source the code.