Realizing a Graphical User Interface for Procedures

Andrew Frederick Cowie
Operational Dynamics

Abstract

Printed procedures have been successfully used to plan and co-ordinate the activities of teams carrying on massive changes and upgrades to mission critical systems, but formatting these documents by hand and working through such events with paper copies has shown some limitations. To improve the experience of using these methods we introduce **xseq**, a graphical user interface for procedures being written in GTK.

Introduction

We have long wanted a way to automate moving through the sorts of procedures we use for mission critical events as described in "Surviving Change" [1]. The thesis of that paper was to document procedures and then to use the practices built up through shared experience to systematically improve the capture and dissemination of knowledge.

People writing down instructions for carrying out tasks is nothing new. Things rapidly become unwieldy, however, when trying to document the tasks that need to be executed by a team for a mission critical change event. There are often multiple participants and the nature of the change is invariably highly complex demanding careful co-ordination. How should this be recorded?

In this paper we introduce **xseq** (so named from a shortening of *XML for Sequences and Series*), a desktop application to allow users to display and collaborate on procedures.

Background

Our industry lacks standards for articulating procedural instructions. In the "Massive Changes and Upgrades" [2] tutorials we presented a style for organizing such procedures, based on our experience of using a specific hierarchy of nested ordered and unordered lists to represent concurrency within an ordered sequence of steps.

Below we have an example of an event conducted during the dot-com era documented using this style, whereby the muppets are about to carry out a significant upgrade of the central database in their production e-commerce site.

Keywords: Procedures, Mission Critical Systems, Changes and Upgrades, XML, Graphical User Interfaces, GTK

A. F. Cowie, "Realizing a Graphical User Interface for Procedures" at *CodeCon 2007*.
Newnes State Forest, NSW: Peter Miller and Company, Sep. 2007.

Production Database Upgrade

Overview

In order to launch the next version of our e-commerce platform, we need to upgrade the schema of the core database at the heart of the application. We also have an outstanding requirement to upgrade the underlying database software, as we have had trouble with several bugs therein which the vendor reports fixed.

Procedure

I. Take site down

Before taking the database offline for its upgrade, we put the site into maintenance mode and safely down the servers. The start time is critical due to expected duration of the database schema upgrade scripts.

1. **Enter maintenance mode**
   - **Fozzie**
     a. Put web site into maintenance mode (load balancer redirect to alternate web servers with static pages)
   - **Gonzo**
     a. Activate IVR maintenance mode
2. **Down services**
   - **Kermit**
     a. Stop all VMs
     b. Stop GFS on database1, database2
     c. Ensure RAID filesystems still mounted
   - **Gonzo**
     a. Stop Apache on web1, web2
3. **Verification**
   - **Kermit**
     a. Verify maintenance mode is active
     b. Verify all VMs down
     c. GO / NO-GO FOR UPGRADE

II. Database work

Run an export of the database in order to ensure we have a good backup prior to upgrading the database software and running the schema change scripts. There is not much concurrent activity here, so those not directly involved in database activity will head for breakfast.

4. **Database safety**
   - **Beaker**
     a. Database to single user mode
     b. Export database to secondary storage
     c. Stop database
   - **Gonzo**
     a. Run out to get coffees for everyone

Figure 1: Excerpt from a procedure used to carry out a mission critical event, in this case upgrading the database of an e-commerce system.

The procedure shows the *steps* that have to occur to achieve the purpose of the event. Within each step, there are one one or more *named* individuals able to work concurrently on their individual *tasks*. Steps are conducted sequentially. Progress moves to the next next step only once all the tasks within the current step are completed.

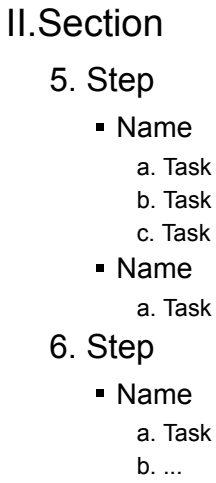

Figure 2: A hierarchical style for documenting procedures.

Sections are a grouping of one or more steps. They represent a stable point; when executing the procedure you can pause at a section boundary, and if something goes wrong they represent the nearest point to rollback to or to push forward to and stop so you can reassess.

This style is a rigid hierarchy where each level has a defined proper name. This facilitates each step of a procedure having a fully qualified name.

Section `III`, Step `2`, `Name`'s Task `d`.

When working through a procedure, the team members active in the current step work through their assigned tasks in order, with each person working independently carrying out their actions at the same time. This is work happening in *parallel*.

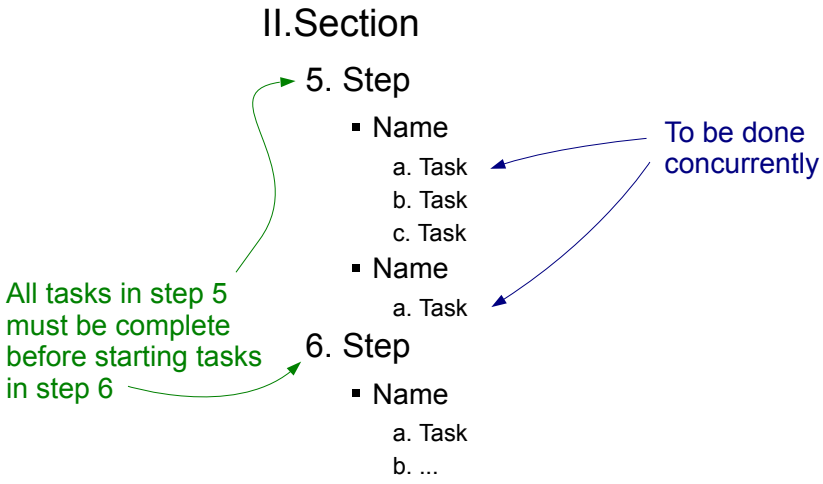

Figure 3: Concurrency within the sequence of steps of a procedure.

In our example scenario, Fozzie, Gonzo, and Kermit are working together through a procedure. We have Fozzie and Gonzo performing step **5**. Gonzo has finished his task in this step but Fozzie is still working on **5 c**, so Gonzo has to wait.

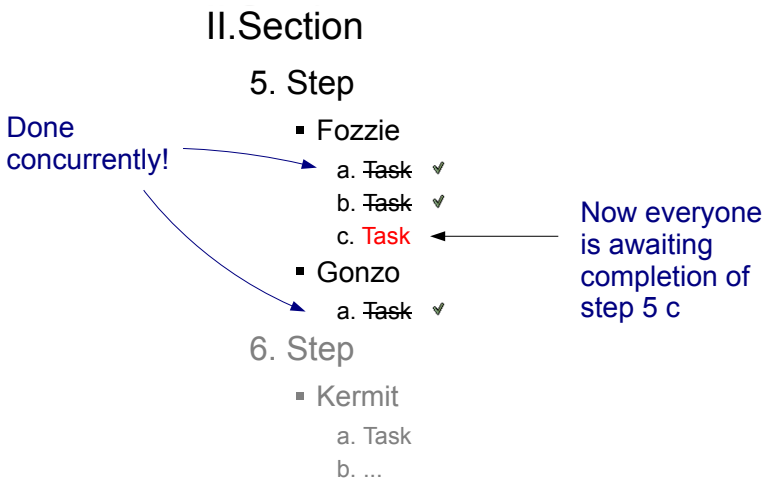

Figure 4: The team must wait until all tasks in a step are complete.

As does Kermit. The key is that the team does not proceed to the next step until all the tasks in the preceding one are complete. Indeed, this is the reason for having steps; they provide a point to co-ordinate progress between disparate participants in an ongoing event. The steps happen in *series*.

With step **5** complete, tasks in step **6** can proceed:

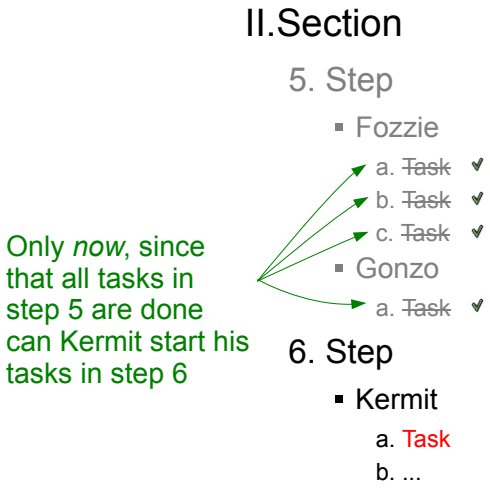

Figure 5: With the step complete, work on the next can begin.

User Interface

Our ambition is to create a desktop software application that would facilitate teams moving through such a procedure. The key goals are summarize the event, be able to show the current status of tasks, and to encourage participants to follow the steps as planned rather than jumping ahead.

Overview Window

One of the problems with long procedures is the amount of detail and information which can make it difficult to see the forest for the trees. One of the recommendations of [2] was to ensure the overall event was summarized so that senior or external audiences could understand the essence of the event without having to read the procedure in detail.

The first innovation in the design of **xseq** was to *automatically* extract the precis paragraphs from each section and to render them as an overview of the procedure.

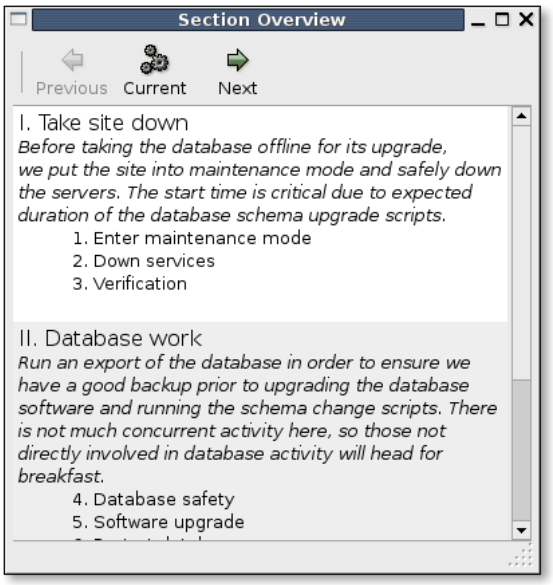

Figure 6: The Overview window, showing the outline of the procedure to be run.

This was our first experience with generating an alternate view of the procedure from the underlying data. Displaying the descriptive text for the section along with the titles of the steps involved provided a powerful tool for getting a high level understanding of the event.

Double-clicking on one of the rows in the GtkTreeView widget causes the DetailsWindow to be raised and focused, and the view switches to showing the details of the selected section.

Details Window

The DetailsWindow is the application's main interface.

From here the user can navigate between sections, see their currently active assigned task highlighted if they have one, and can indicate their status as they work on their assigned task by using buttons corresponding to those different statuses.

The navigation buttons follow the same visual convention as the `OverviewWindow`, allowing the user to look ahead in the procedure to see what's coming.

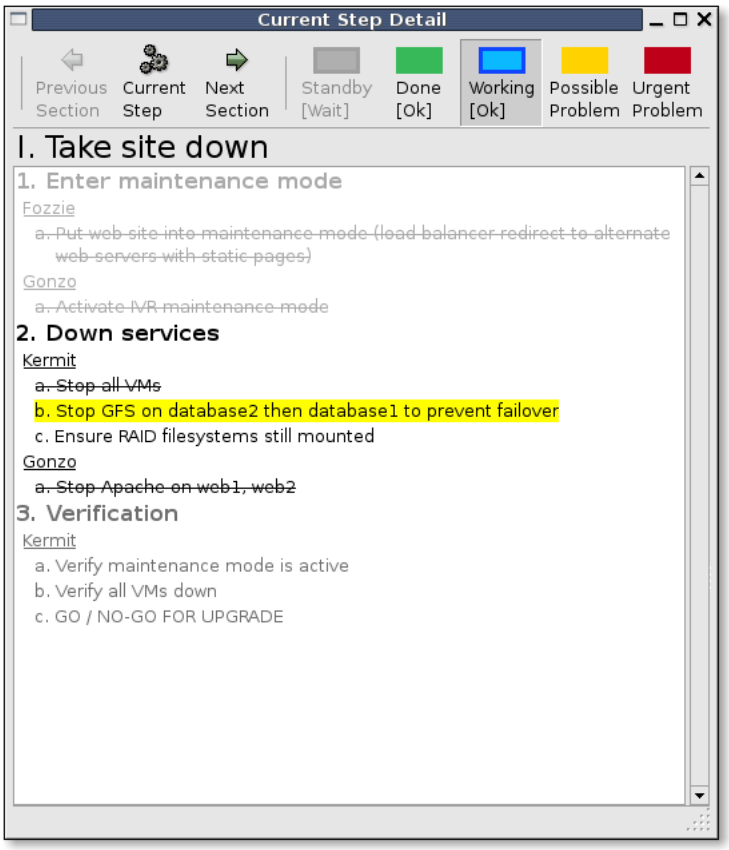

Figure 7: The Details window.

By default the status lights are either gray, representing the user being `Idle` because they have no currently assigned task, or blue `Working` indicating they currently have an active task and work is underway.

Pressing the green `Done` button marks the task as complete and advances the user to the next task. In future pressing any of the other status buttons will signal an update to the event controller, facilitating faster communication in the team and helping raise situational awareness across the team.

Testing and Development

Developing GTK graphical user interface code can be tedious and repetitive. Frequent iteration is required to work out placement of widgets and other graphical elements, and each iteration requires restarting the program. To facilitate testing, several helper windows are initialized when starting the program.

Loading Helper

When developing the user interface code a procedure needs to be first selected and loaded into the program. In production this will be done via message over the network, but for now a file can be specified from the local hard drive:

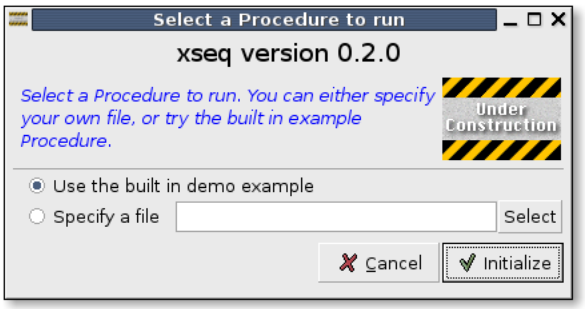

Figure 8: The Test Loading window, used when starting the program to load a procedure for testing and development.

Control Helper

Once loaded, we need to execute through the procedure. The DetailsWindow's buttons record when the user facing the program completes a task, but we need to simulate other participants in the event executing their tasks, and so a TestControl window is provided. It allows the developer to pick which person is the active user and then provides buttons to artificially advance the other participants' tasks.

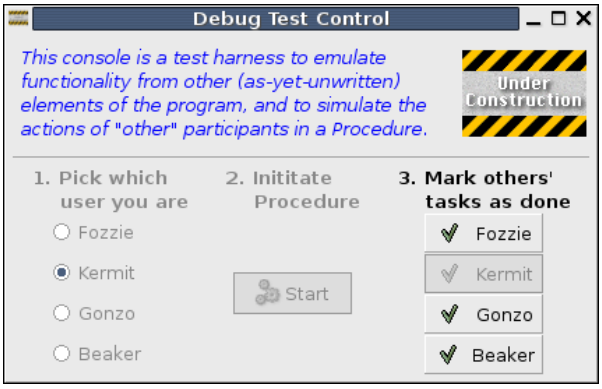

Figure 9: The Test Control window, used to manually advancing through a procedure when debugging.

Communicating Status

One of the lessons arising our experience applying procedures to organize mission critical events has been the importance of communicating status.

We have introduced *status colours* to allow participants in a procedure to communicate their current state as they work through a task.

This was originally inspired by NASA mission control during the Apollo era. [3] The flight controllers had toggle switches used during launch to indicate their status:

Green	Everything is ok.
Yellow	I am working a potential issue. Call me when you get a moment.
Red	I need to speak to the flight director, *now*. There is a serious problem and I am considering calling a launch abort within the next two seconds.

At first we thought we'd only need `Ok` and `Problem`. There were a few problems with this. Firstly, we have found that people often forget to tell anyone that they have finished a task, being heads-down

and focused on moving to the next step or simply being oblivious that other people would need to know. So we needed to add a `Done` status.

Secondly, in our experiments with procedures we have found that people are often "reluctant to pull the fire alarm". This is either because they don't want to admit there's a problem (a surprisingly common situation when highly motivated techies with strong egos are involved), or because they don't perceive that what they are trying to solve warrants raising the alarm. Programmers and sysadmins spend most of their working lives dealing with one tricky problem after another so sometimes can under-appreciate that problems encountered during time-critical events can be more serious. To avoid this, we created a "possible problem" status labelled as `Problem` and "urgent problem" status labelled as `Critical`. Functionally there's not much difference between them but they allow the person responsible for an event to have a better chance of getting early warning of a potential issue.

Finally we needed to communicate the difference between working on a task and not currently working on a task. Sometimes you're on standby waiting for other members of the team to do their tasks. We assigned that to dark gray, following the user interface convention of "graying out" options that are not currently available.

Gray	Standby, no task currently assigned to me. `Idle`.
Green	Task is complete, waiting for next step where I have a task to do. `Done`.
Blue	Task is currently being worked on `Working`.
Yellow	There is a possible issue; it's not going as fast as it should or something is wrong, `Problem`.
Red	Serious issue has been encountered, `Critical`!

This is realized in **xseq** in prominent toolbar buttons in the application's main DetailsWindow (Figure 7) as previously shown, and as a small "always-on-top" decoration-less window that is placed bottom right of screen (Figure 10).

Figure 10: The QuickButtons status popup.

Often when working on a mission critical event participants will be in a full screen terminal or web browser window. Keeping the status QuickButtons on the screen gives the user a way to get back to the procedure and serves as a very strong reminder that they need to communicate status as they work. Easy enough to click the green `Done` button, bringing the procedure window back to the foreground, and refocusing the user on the next task.

File format for procedures

Binary formats like Word documents are unsuitable for storing procedures. [2] Online systems like wikis suffer from other problems. We want a way to write a procedure to disk so they can be interchanged and history kept in a version control system. This implies a text format with a known and enforceable schema.

XML is a textual file format that allows documents containing structured hierarchical data to be compared against a schema to ensure the contents are both well-formed (correct syntax) and valid (actually conforming to the requirements of the specified file format).

Schema description

In **xseq**, procedures are written in XML according to the following doctype schema:

```
<!ELEMENT procedure (title,overview?,section*)>
<!ATTLIST procedure id ID #IMPLIED>
<!ATTLIST procedure xmlns CDATA #FIXED "http://namespace.operationaldynamics.com/procedures/0.4">

<!ELEMENT title (#PCDATA)>
<!ATTLIST title id ID #IMPLIED>

<!ELEMENT overview (#PCDATA)>
<!ATTLIST overview id ID #IMPLIED>


<!ELEMENT section (title, precis?,step+)>
<!ATTLIST section start CDATA #IMPLIED>
<!ATTLIST section id ID #IMPLIED>

<!ELEMENT precis (#PCDATA)>
<!ATTLIST precis id ID #IMPLIED>


<!ELEMENT step (title, name+)>
<!ATTLIST step id ID #IMPLIED>

<!ELEMENT name (task+)>
<!ATTLIST name who CDATA #REQUIRED>
<!ATTLIST name id ID #IMPLIED>

<!ELEMENT task (#PCDATA)>
<!ATTLIST task id ID #IMPLIED>
<!ATTLIST task status (done | working | problem | critical) #IMPLIED>
```

Listing 1: XML DTD schema for a procedure

Use of whitespace

The screenshots shown above were of the application running through the a sample procedure written to this schema (Listing 2 below).

Though optional in XML, the whitespace and indentation used clearly shows the hierarchical relationship between `sections`, `steps`, `names` and `tasks`. You can also see preliminary attempts to supply additional metadata, for example the expected start time.

Challenges

There are a number of critiques of this format as it presently stands. The title of the procedure is a full element whereas the title of each step is merely an attribute string. Not having the step title as a distinct element limits expressiveness.

The titles, overview, precis, and task bodies are all plain text. The opportunity for using a richer markup would seem to beckon; potentially this could be achieved using the extensibility of XML via the namespace mechanism to map in rich text expressed as valid XHTML, but we would somehow have to limit this to a strict subset we would be able to support and would require a far more advanced renderer than the GtkTreeView and GtkTextView widgets being used presently.

Finally, although the ability to check well-formedness and to validate a document is the strength of XML, the authoring experience is not ideal. Writing element tags and attempting to maintain indentation by hand is tedious and error-prone.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE procedure PUBLIC "-" "file:///home/andrew/workspace/XmlSequence/web/procedure.dtd">
<procedure>
    <title>Production Database Upgrade</title>
    <overview>
        In order to launch the next version of our e-commerce platform, we need
        to upgrade the schema of the core database at the heart of the
        application. We also have an outstanding requirement to upgrade the
        underlying database software, as we have had trouble with several bugs
        therein which the vendor reports fixed.
    </overview>
    <section title="Take site down" start="0515">
        <precis>
            Before taking the database offline for its upgrade, we put the site
            into maintenance mode and safely down the servers.  The start time
            is critical due to expected duration of the database schema upgrade
            scripts.
        </precis>
        <step title="Enter maintenance mode">
            <name who="Fozzie">
                <task>
                    Put web site into maintenance mode (load balancer redirect
                    to alternate web servers with static pages)
                </task>
            </name>
            <name who="Gonzo">
                <task>Activate IVR maintenance mode</task>
            </name>
        </step>
        <step title="Down services">
            <name who="Kermet">
                <task>Stop all VMs</task>
                <task>Stop GFS on database1, database2</task>
                <task>Ensure RAID filesystems still mounted</task>
            </name>
            <name who="Gonzo">
                <task>Stop Apache on web1, web2</task>
            </name>
        </step>
        <step title="Verification">
            <name who="Kermet">
                <task>Verify maintenance mode is active</task>
                <task>Verify all VMs down</task>
                <task>GO / NO-GO FOR UPGRADE</task>
            </name>
        </step>
    </section>
    <section title="Database work">
        <precis>
            Run an export of the database in order to ensure we have a good
            backup prior to upgrading the database software and running the
            schema change scripts. There is not much concurrent activity here,
            so those not directly involved in database
            activity will head for breakfast.
        </precis>
        <step title="Database safety">
            <name who="Beaker">
                <task>Database to single user mode</task>
                <task>Export database to secondary storage</task>
                <task>Stop database</task>
            </name>
            <name who="Gonzo">
```

Listing 2: Example procedure showing XML file structure

Future Work

The first version of **xseq** was 0.2.0, released 2 May 2005. [4]. It is implemented using the **java-gnome** 4.0 language bindings [5] to the GTK and GNOME graphical user interface and desktop environment libraries on Linux. Ongoing development is working towards a 0.4 release. You can see the program in action (Figure 11) below.

Future lines of development will include:

Networking

We are currently investigating Jabber/XMPP using the Telepathy framework [6] as the network transport layer. Originally conceived of as an instant messaging protocol, XMPP extends this by provides a binding of an application to a server on behalf of a user and a way of addressing messages to a specific endpoint at that server. We envision that the **xseq** program could raise itself as a node in the XMPP network and communicate task status to other nodes in a peer-to-peer fashion.

Editor

Right now procedures are being written manually in XML in text files. Writing XML by hand is tedious and error prone. The ability to create new procedures in the GTK app is obviously needed.

Procedures often need to change as they are being run. Once we have an editor the notion of being able to update the tasks to be done as a procedure is being executed live might be within reach.

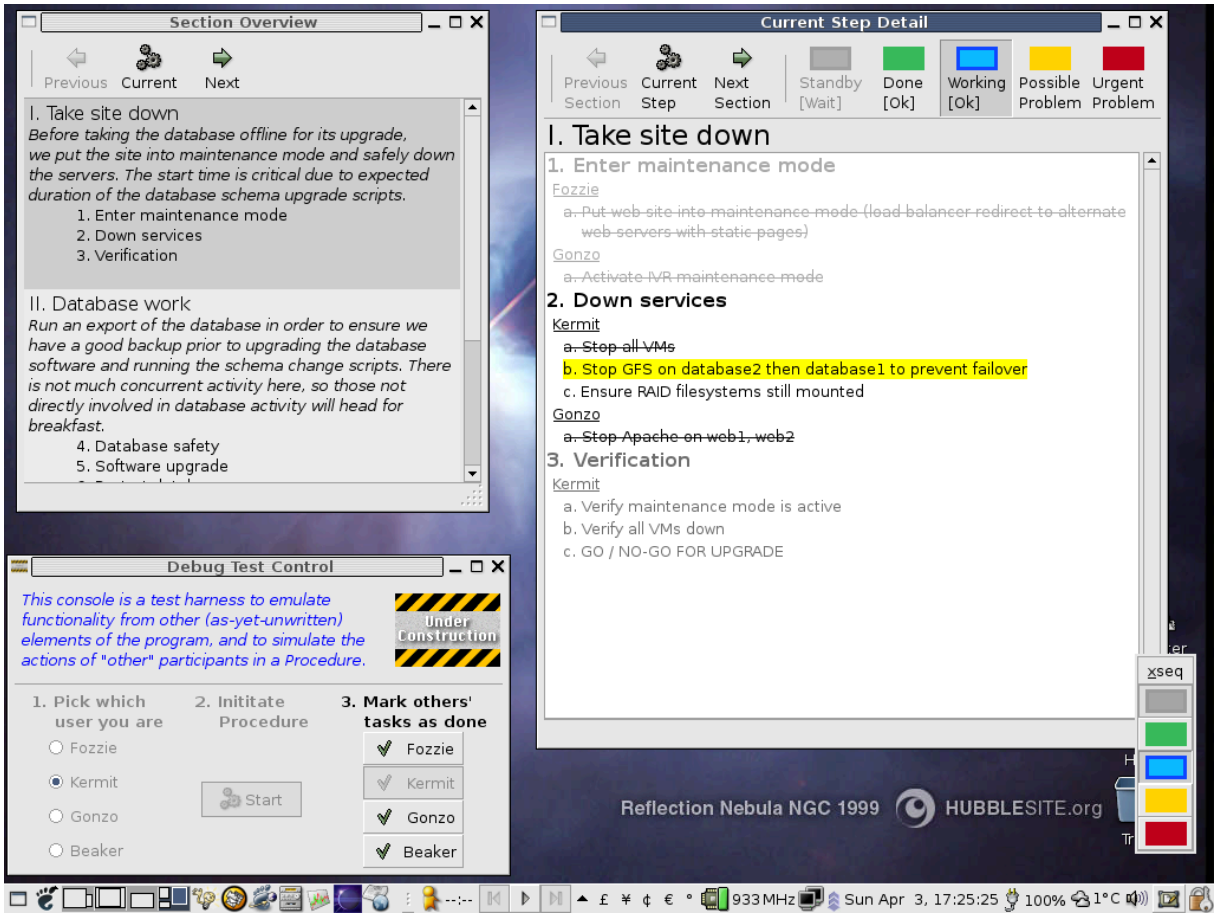

Figure 11: Screenshot of a desktop with **xseq** running in test mode.

References

[1] A. F. Cowie, “Surviving Change,” in *Systems Administrator Guild of Australia Annual Conference Proceedings*, SAGE-AU, Aug. 2004. doi: 10.31224/5596.

[2] A. F. Cowie, “Massive Changes and Upgrades,” in *Proceedings of the 19th Conference on Systems Administration (LISA '05)*, D. N. Blank-Edelman, Ed., USENIX, 2005.

[3] G. Kranz, *Failure Is Not an Option*. Berkley, 2000.

[4] A. F. Cowie, “xseq 0.2.0 released!,” Operations and Other Mysteries. Accessed: Sep. 15, 2007. [Online]. Available: https://blogs.operationaldynamics.com/andrew/software/xseq/xseq-0-2-0-released

[5] A. F. Cowie *et al.*, *java-gnome User Interface Library*. The GNOME Project. Accessed: Sep. 15, 2007. [Online]. Available: https://java-gnome.sourceforge.net/

[6] R. Burton *et al.*, *Telepathy Framework*. Freedesktop Project. Accessed: Aug. 28, 2007. [Online]. Available: https://telepathy.freedesktop.org/